**Novo Curso de Javascript Completo, Profissional e Moderno - Curso de Javascript Moderno - Aula 01**

Javascript e uma linguagem que roda tanto do lado do cliente quanto do lado do servidor.

Devemos baixar o node, podemos instalar o nodeJs no próprio site deles.

Para conferir se o nodeJs está instalado podemos escrever no prompt de comando **Node -v**

Para começar a escrever nosso codigo em Javascript devemos criar um arquivo com .js no final do nome, o nome pode ser qualquer um por exemplo teste.js desde que tenha .js no final.

Javascript e uma linguagem interpretada e não compilada significa que é executada em tempo real não preciso compilar o codigo e criar um executável.

.

## **Console.log**

Faz a impressão no console, e o comando de saída, tudo que for escrito nele e imprimido no console.

console.log ("Alo, mundo!");

Esse codigo vai imprimir no console o texto Alo, mundo!

.

## **Comentários**

Os comentários em um codigo podem servir para diversas coisas como por exemplo explicar para que serve um codigo ou até mesmo organizar, os comentários não serão rodados ou executados nos codigo e apenas algo visual para quem está escrevendo no codigo, resumindo não vai mudar nada no resultado.

Para fazer um comentário em js existe duas maneiras, se fizermos o comentário em uma linha somente nos colocamos duas barras ( // ) e o texto logo em seguida,

// Comentário de uma linha

.

mas se quisermos comentar um bloco de codigo com várias linhas podemos colocar o codigo entre uma aspa e um asterisco (/\* comentário aqui \*/).

/\*

    Esse

    e um comentário

    Em Js

\*/

.

## **Como rodar Javascript dentro do browser?**

Para rodar o Javascript no browser devemos unir o Javascript com o html, entao devemos criar um arquivo .html, e chamar o Javascript dentro do HTML, depois de criar o arquivo .html devemos escrever o codigo base do html

<!DOCTYPE *html*>

<html *lang*="pt-br">

<head>

    <meta *charset*="UTF-8">

    <meta *name*="viewport" *content*="width=device-width, initial-scale=1.0">

    <tile>Aula 1 de JavaScript-CFBCursos</titile>

</head>

<body>

</body>

</html>

E dentro da área de body que é o corpo do site escrevemos

    <script *src*="aula01.js"></script>

Aqui nós chamamos o arquivo .js para dentro do html e quando executamos o html e aberto o navegador junto com o Javascript que já está sendo executado.

E importante que coloquemos a tag script dentro do body e no final de todo codigo que é escrito dentro do body, pois quando a gente for escrever DOM que é a manipulação de elementos em Javascript logo veremos isso, e preciso que o html seja carregado antes do Js.

.

# Javascript em Modo Estrito, você sabe o que é? - Curso de Javascript Moderno - Aula 02

## **Modo estrito**

Serve para deixar nosso codigo mais limpo com menos sujeira, com menos coisas inutilizadas, por exemplo no modo estrito não conseguimos usar uma variável que não foi declarada, e as variáveis criadas tem que ser usadas senão do erro, entao erros que poderiam ser executados sem o modo estrito passam a ser necessários correção, o modo estrito não deixa passar nenhuma falha no codigo.

Para ativar o modo estrito nos colocamos no codigo

"use strict"

Que já será ativado,

.

**Diferença entre LET, VAR e CONST em Javascript - Curso de Javascript Moderno - Aula 03**

## **Oque e uma variável**

E uma posição dentro da memória ram, quando nós estamos criando uma variável nós estamos basicamente criando um espaço dentro da memória ram e nomeamos esse espaço onde podemos colocar algum dado dentro desse espaço.

Pense em um armário com 8 gavetas onde temos a gavetas enumerada tipo gaveta 1, gaveta2 e por aí em diante, cada gaveta dessa e um espaço no armário onde podemos guardar materiais e a mesa logica da variável onde o armário e a memória ram em geral e as gavetas são as variáveis que são espaços vazios para guardar coisas, e o materiais são os dados que serão guardados.

.

## **Declarando uma variável**

var nome = "John"

Esta e uma variável, primeiro declaramos a variável usando VAR e o nome do espaço na nossa memória Ram (variável) se chama nome e depois utilizamos o sinal de = (igual) que no Javascript tem o sentido de atribuir entao a palavra “John” (string) e colocada dentro da variável nome.

Outra coisa e que para usar a variável ela deve estar acima no codigo para podermos utilizar lá em baixo, entao primeiro ela precisa ser declarada para depois utilizarmos pois na hora da execução de codigo, o navegador lê o codigo de cima para baixo, entao você concorda que precisa existir a variável antes de podermos usar ela.

E se por exemplo quisermos usar a variável lá no final do codigo ou em qualquer lugar, a variável não pode ser declarada dentro por exemplo de uma function, ou em uma condicional que são fechadas por {} (chaves), pois a variável será declarada somente lá dentro e para usar a variável fora das chaves e impossível pois lá ela não existe.

Podemos declarar uma variável de 3 maneiras

### **1 maneira**

let num1=10

let num2=10

### **2 maneiras**

let num1=10, let num2=10

Dessa maneira poderemos declarar várias variáveis em uma linha so separadas por virgulas.

### **3 maneiras**

let num1=num2=num3=10

Aqui declaramos 3 variáveis com o valor 10 sendo que, a variável recebe o número 10 aí a variável num2 receber o valor da variável num3 e a variável num1 recebe o calor da variável num2.

.

## **Diferença entre Let, Const e Var**

Resumido declarando a variável com **let** se colocarmos dentro de uma condicional por exemplo a variável so poderá ser usada lá dentro e não fora, mas agora se usar Var ela poderá ser usada somente dentro e fora dessa condicional, mas agora se a condicional estiver dentro de uma function o var e o let não poderão ser usados fora da function.

**Var** = “Se for declarada dentro de um escopo global poderá ser utilizado por todo o escopo, mas se declarada dentro de um escopo local, poderá ser utilizado apenas nele.“

**Let** = “Se for declarada dentro de um escopo global poderá ser utilizado por todo o escopo, e se for declarada dentro de um escopo local poderá ser utilizado dentro dele todo e uma camada fora dele, mas se um escopo local estiver dentro de um escopo local ele poderia sair de apenas um escopo e será utilizado lá dentro tambem.”

**Const** = “Quando a variável for declarada com algum dado, o valor dessa variável depois não poderá ser mudado, o único dado que vai ficar e o que foi atribuído quando a variável foi declarada, resumindo não podemos atribuir um novo valor na variável declarada com const”

.

## **Mudança entre tipo de dados em variáveis e no conteúdo**

Podemos mudar algo que está dentro da variável apenas atribuindo seu respectivo dado, entao se já declaramos a variável antes com algum valor, se quisermos mudar o conteúdo dessa variável podemos apenas atribuir um outro valor a ela.

let nome = "JohnChk"

nome = 10

console.log(nome)

O resultado que será imprimido vai ser o número 10, pois ele foi atribuído depois da criação da variável e uma variável cabe apenas um dado em seu espaço.

.

# Aprenda sobre os Operadores Matemáticos em Javascript - Curso de Javascript Moderno - Aula 04

## **Operadores aritméticos**

## **+** (Operador de soma)

## **-** (subtração)

## **/** (divisão)

## **\*** (multiplicação)

## **%** (resto de divisão)

## **++** (incremento)

## **--** (decremento)

## **+=** (incremento)

## **-=** (decremento)

A precedência de operadores conta a mesma regra da matemática tradicional.

.

## **Operações exemplos**

Num1=5

num2=10

res=num1+num2

console.log(res)

Aqui a variável num1 tem o valor 5 e a variável nukm2 tem o valor 10, aí pegamos a variável res e jogamos dentro dela o resultado da soma de num1 + num2 que será 15.

.

## **Podemos fazer tambem dentro do console.log()**

num1=5

num2=10

console.log(num1-num2)

Aqui nos subtraímos num1-num2 e será mostrado a resposta no terminal que e onde o console.log mostra quando imprime o resultado será 5.

.

## **Ordem de precedência**

num1=5

num2=10

console.log(num1-num2)\*2

nesse caso pensando na ordem de precedência primeiro será num1-num2 por causa dos parênteses, e depois o resultado será multiplicado por 2 assim mostrando o valor deste calculo que e 10.

.

## **Resto da divisão**

num1=15

num2=2

res=num1%num2

console.log(res)

Aqui estamos dividindo com % que na verdade não mostrara o resultado da divisão, mas sim o resto dessa divisão, entao 15 dividido por 2 da 7 mas resta 1, oque será mostrado no console.log será 1 pois é o resto.

.

## **Incremento**

num1=10

num1++

num1++

Aqui nesse caso nossa variável num1 tem o valor 10, mas o incremento sempre adiciona um numero a essa variável entao quando colocamos o nome da variável que é num1 e colocamos ++ será adicionado um numero, entao como fizemos 2 vezes o valor final da variável num1 será 12.

.

## **decremento**

num1=10

num1--

num1--

Aqui nesse caso nossa variável num1 tem o valor 10, mas o incremento sempre retira um número dessa variável entao quando colocamos o nome da variável que é num1 e colocamos -- será retirado um numero, entao como fizemos 2 vezes o valor final da variável num1 será 8.

.

## **Incremento PRO**

Tem a mesma função do incremento so que podemos escolher o valor que queremos incrementar, quanto no incremento comum podemos somar apenas 1 a variável aqui no podemos escolher quanto queremos incrementar.

num1 = 10;

num1+= 5;

neste exemplo será incrementado o valor 5 a variável num1 ficando assim 15, ele estará adicionando mais um valor, podemos incrementar outras variáveis com algum valor

num1 = 10;

num2 = 3

num1+= num2;

O resultado será 13 pois o valor de num1 era 10 aí foi incrementado o valor de num2 ficando assim o resultado final 13.

.

## **Decremento PRO**

Tem a mesma função do decremento comum so que podemos escolher o valor que queremos decrementar, enquanto no decremento comum podemos subtrair apenas 1 a variável aqui nós podemos escolher quanto queremos decrementar.

num1 = 10;

num1-= 5;

neste exemplo será decrementado o valor 5 a variável num1 ficando assim 5, ele estará retirando mais um valor, podemos tambem decrementar outras variáveis com algum valor

num1 = 10;

num2 = 3

num1-= num2;

O resultado será 7 pois o valor de num1 era 10 aí foi decrementado o valor de num2 ficando assim o resultado final 7.

.

# Operadores Relacionais em Javascript, aprenda como usar! - Curso de Javascript Moderno - Aula 05

## **Oque e uma operação relacional?**

Operação relacional e uma comparação usaremos operadores de comparação

## **Operadores**

### **> Maior**

let num1=10

let num2=5

let num3=10

console.log(num1 > num2)

Aqui estamos perguntando no console.log se num1 que tem 10 atribuído a ele é maior que num2 que tem o número 5 atribuído a ele, o resultado será verdadeiro (true) pois num1 e maior que num2.

.

### **>= Maior ou igual**

let num1=10

let num2=5

let num3=10

console.log(num1 >= num3)

Aqui estão perguntando se num 1 e maior ou igual a num3, entao se for maior ou igual o resultado será verdadeiro(true) senão se não for o resultado será falso(false).

O resultado será verdadeiro(true) pois num1 e igual a num3 mesmo que não seja maior, pode ser qualquer uma das opções, se for maior ou igual.

.

### **< Menor**

let num1=10

let num2=5

let num3=10

console.log(num1 < num2)

Aqui estamos perguntando no console.log se num1 que tem 10 atribuído é menor que num2 que tem o número 5 atribuído a ele, o resultado será falso (false) pois num1 não é menor que num2.

.

### **<= Menor ou igual**

let num1=10,let num2=5,let num3=10

console.log(num1 <= num3)

Aqui estão perguntando se num1 e menor ou igual a num3, entao se for menor ou igual o resultado será verdadeiro(true) senão se não for o resultado será falso(false).

O resultado será verdadeiro(true) pois num1 e igual a num3 mesmo que não seja menor, pode ser qualquer uma das opções, se for menor ou igual.

.

### **== igual**

let num1=10,let num2=5,let num3=10

console.log(num1 == num3)

Aqui estamos perguntado se num1 e igual a num3, o resultado será verdadeiro (true) pois num1 e igual a num3.

.

### **= Atribuição**

Atribuição nós já vimos antes e já usamos, usamos ao atribuir um valor(dado) a uma variável.

let num1=10

Neste exemplo estamos atribuindo o valor(dado) número 10 a variável nomeada como num1.

.

### **!() not**

Esse e um pouco diferente dos demais pois ele inverte os valores

let num1=10, let num3=10

console.log(num1 == num3)

por exemplo nesta comparação o valor será verdadeiro(true) pois num1 e igual a num3, mas aí nós adicionamos o sinal NOT a ele.

console.log(!(num1 == num3))

nós adicionamos e colocamos os valores entre colchetes, o sinal de NOT(!( )) ele nega, entao se o valor deu verdadeiro(true), será inverso para false, e se o valor der false será invertido para true, ele simplesmente inverte o resultado.

.

### **!=**

let num1=10

let num2=5

let num3=10

console.log(num1 != num3)

Aqui estamos perguntado se num1 e diferente (!=) de num3 e o resultado e obviamente falso(false), pois num1 e num2 tem o mesmo valor.

.

# Respondendo Perguntas dos Inscritos Parte 1 (aulas de 1 a 5) - Curso de Javascript Moderno - Aula 06

## **Qual a diferença entre == e ===**

O == faz comparação do valor da variável e apenas isso, uma comparação rasa.

O === Faz uma comparação mais profundo pegando tambem o tipo de dados, se é uma string, um número entre outras coisas, entao além de comparar os valores ele compara o tipo de dados e tambem o lugar na memória.

Temos tambem a seguinte questão

let v1={nome:"Bruno"}

let v2={nome:"Bruno"}

console.log(v1===v2)

Por mais que os valores sejam o mesmo e a variável tenha o mesmo tipo, eles não ocupam o mesmo lugar na memória entao não são iguais.

O nome da variável não importa

.

## **Trabalhando pelo prompt de comando**

Podemos fazer isso colocando o caminho da nossa pasta no cmd, mas temos que verificar que diretório estamos usando pode ser que seja o c ou o d, caso nossa pasta esteja no diretório d nos apenas colocamos d: se nossa pasta com o projeto estiver no diretório c nos colocamos c: e depois escrevemos node (nome do arquivo) sem os parênteses.

.

## **Diferença entre Diferente e NOT**

O sinal de diferente != ele confere se algo e diferente do outro, se for diferente da verdadeiro(true pois é uma verdade mas o NOT ele simplesmente pega a comparação e inverte o resultado, entao se uma comparação der verdadeiro o NOT converte para falso (false).

.

## **Codigo em JS para descobrir em qual dispositivos estamos usando na pagina web**

if(navigator.userAgent.match(/Android/i)

|| navigator.userAgent.match(/WebOS/i)

|| navigator.userAgent.match(/iPhone|iPad|iPod/i)

|| navigator.userAgent.match(/BlackBerry/i)

|| navigator.userAgent.match(/Windows Phone/i)

|| navigator.userAgent.match(/Opera Mini/i)

|| navigator.userAgent.match(/IEMobile/i)

){

    console.log("Celular")

}else{

    console.log("PC")

}

.

## **Oque eo DOM (manipulação de dados)**

E a arvore de elementos dentro do html, temos uma estrutura que são os elementos da nossa pagina, o DOM ele manipula os elementos identificando a arvore de elementos no navegador.

Ele cria essa arvore de elementos (DOM) e possibilita que o JavaScript o manipule.

.

## **Defer**

Quando colocamos a propriedade defer

    <script *src*="aula06.js" *defer*></script>

Ele faz com que de uma preferencia para que o HTML seja carregado primeiro para depois o JavaScript seja lido, entao podemos deixar esse codigo para chamar o javaScript la no header invez de la embaixo como e normalmente posto, o resultado eo mesmo, mas a prefrencia e que seja posto la em baixo ainda.

.

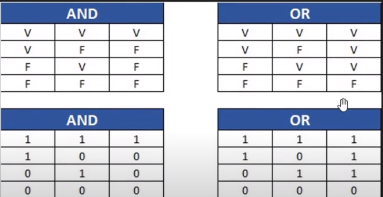
# Operadores Lógicos em Javascript, aprendendo a usar! - Curso de Javascript Moderno - Aula 07

**&&** -> and -> e

**||** -> or - > ou

**!() ->** not -> não

.



Essa tabela nos ajudara,

.

## **&& (E lógico):**

O operador && é usado para verificar se duas condições são ambas verdadeiras. Ele retorna true somente se ambos os lados da expressão forem avaliados como verdadeiros. Se pelo menos um dos lados for falso, o operador && retorna false.

let a = true;

let b = false;

console.log(a && b); // Retorna false, porque a é verdadeiro mas b é falso

console.log(a && true); // Retorna true, porque ambos a e true são verdadeiros

.

## **|| (OU lógico):**

O operador || é usado para verificar se pelo menos uma das condições é verdadeira. Ele retorna true se pelo menos um dos lados da expressão for avaliado como verdadeiro. Somente quando ambos os lados forem falsos, o operador || retorna false.

let a = true;

let b = false;

console.log(a || b); // Retorna true, porque a é verdadeiro, mesmo que b seja falso

console.log(false || false); // Retorna false, porque ambos os lados são falsos

.

## **! (Negação lógica):**

O operador ! é usado para negar o valor de uma expressão booleana. Ele inverte o valor de verdadeiro para falso e de falso para verdadeiro.

let a = true;

console.log(!a); // Retorna false, porque a é verdadeiro e a negação é falsa

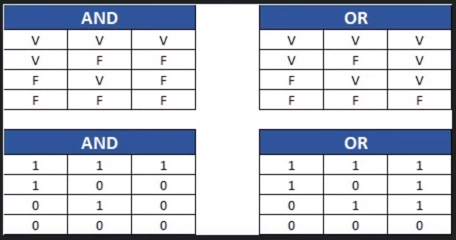
let b = false;

console.log(!b); // Retorna true, porque b é falso e a negação é verdadeira

.

]

# Operadores Bitwise em Javascript - Curso de Javascript Moderno - Aula 08



O BitWise nos operamos os bits.

E um pouco complicado de entender, usaremos o numero 10 e 11

let n1=10; /\*10 em binario (1010) \*/

let n2=11; /\*11 em binario (1011) \*/

let res = n1 & n2

console.log(res) /\* resultado sera 10 \*/

temos o codigo acima e o resultado acima sera 10, mas porque?

Se pegarmos os códigos binários as casas dos bits que são equivalentes eles irão se repetir por causa do **&**, pois como já vimos anteriormente o & exige que os dois seja equivalentes, entao pegamos o primeiro numero em binário do 10 e do 11,

(1010)

(1011)

O primeiro numero de ambos e 1 e eles se repetem, entao o retorno sera 1, o segundo numero de ambos e 0 e se repetem entao o resultado sera 0, o terceiro numero de ambos e 1 entao o retorno sera 1, o 4 numero de ambos eles não se repetem pois temos o numero 1 e o numero 0 entao o retorno sera 0 (quando não se repete o retorno sera 0) ficando assim (1010).

E o codigo em binário do numero 10 e (1010) entao por isso o resultado la no nosso codigo foi 10.

let res = n1 & n2

console.log(res) /\* resultado sera 10 \*/

.

## **mas agora fazendo o mesmo exemplo com || (ou)**

let n1=10; /\*10 em binario (1010) \*/

let n2=11; /\*11 em binario (1011) \*/

let res = n1 || n2

console.log(res) /\* resultado sera 11 \*/

temos o codigo acima e o resultado acima sera 11, mas porque?

Se pegarmos os códigos binários as casas dos bits o || falara que pode ser um ou outro os dois não prescisam ser equivalentes igual o &, pois como já vimos anteriormente o || não exige que os dois seja equivalentes mas pode ser um ou outro, entao pegamos o primeiro numero em binário do 10 e do 11,

(1010)

(1011)

O primeiro numero de ambos e 1 e eles se repetem mas tanto faz que pode se repetir ou nao, entao o retorno sera 1, o segundo numero de ambos e 0 e se repetem entao o resultado sera 0, o terceiro numero de ambos e 1 entao o retorno sera 1, o 4 numero de ambos eles não se repetem pois temos o numero 1 e o numero 0 mas o || não se importa porque pode ser um ou outro entao o retorno sera 1 (quando não se repete o retorno sera 1) ficando assim (1011).

E o codigo em binário do numero 11 e (1011) entao por isso o resultado la no nosso codigo foi 11.

let res = n1 || n2

console.log(res) /\* resultado sera 11 \*/

.

## **mas agora fazendo o mesmo exemplo com ^ ()**

let n1=10; /\*10 em binario (1010) \*/

let n2=11; /\*11 em binario (1011) \*/

let res = n1 ^ n2

console.log(res) /\* resultado sera 1 \*/

Nesse caso eo mais simples, ele retorna 0 aos bits que se repetem e 1 aos que não se repetem

(1010) -> 10

(1011) -> 11

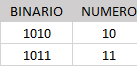
No nosso exemplo aqui os três primeiros se repetem entao o resultado sera 0 e o ultimo não se repete entao o resultado final e 1.

.

## **Operações de deslocamento de BIT**

Cara so falar que isso e incrível.

Lembrando os capítulos passados nos vimos sobre os binários e como eles se comportam diante de operadores lógicos, mas agora aprenderemos sobre deslocação de bit, e pegar um bit e modificar ele em JavaScript para nos trazer outro resultado.



Mostrando em uma planilha do excel podemos ver dois números e seus respecticvos códigos em binário, mas oque aconteceria se deslocássemos o codigo em biinario uma casa para a esquerda, você sabe?

Isso simplesmente faria o codigo binário pular uma casa para a esquerda e o numero 0 preencheria o local vazio ficando assim



Nos pulamos uma casa para a esquerda e sempre que um lugar fica vazio e preenchido com 0.

let n1=10; /\*10 em binario (1010) \*/

let res = n1 << 1

console.log(res) /\*O resultado sera 20\*/

Em JavaScript o codigo ficaria assim, pulamos 1 casa para esquerda o numero 10 que esta dentro da variável n1 e depois jogamos o resultado dentro da variável res e mostramos no console, o resultado assim ficando 20 pois o codigo em binário de 20 e (10100).

Mas podemos deslocar para a direta tambem.

let n1=10; /\*10 em binario (1010) \*/

let res = n1 >> 1

console.log(res) /\*O resultado sera 5\*/

Tem a mesma logica do deslocamento para a esquerda so que o codigo agora sera (101\_ que da o numero 5 pois foi deslocado o codigo em binário (1010) para a direita e quando deslocamos estamos apagando o ultimo zero.

**IMPORTANTE: o deslocamento com apenas 1 numero e apenas um exemplo podemos deslocar 2 casas ou ate mais tambem tanto para a direita quanto para a esquerda resultando em um novo numero.**

## **Onde poderemos usar isto?**

Caso você não tenha percebido, sempre que estamos deslocando um bit de algum codigo para a esquerda o numero dobra, e a mesma coisa de multiplicar por 2, entao poderemos usar desta maneira, a mesma coisa deslocando para a direita.

.

# Diferença entre Pré Incremento e Pós Incremento - Curso de Javascript Moderno - Aula 09

O incremento no JavaScript e uma maneira de adicionar um numeroa uma variável, e como se fosse num = num + 1, entao se por exemplo temos uma variável com o numero 10 e nos fazemos um incremento, logo ele sera 11 e se depois incrementarmos denovo ele sera 12

Abaixo temos um exemplo de codigo.

let n=10

n++

console.log(n)

No codigo acima tivemos um exemplo de incremento onde o resultado sera 11, mas o incremento pode se dividir em duas opções que eo pre incremento e o pos incremento

### **Pre Incremento abaixo:**

let n=10

console.log(n++)

Ele primeiro mostra o numero que já esta na variável e depois incrementa um numero a ela, o que seria imprimido primeiro seria o numero 10

### **pós incremento abaixo:**

let n=10

console.log(++n)

Ela primeiro incrementa um numero a variável e depois mostra o valor do numero atual, o que seria imprimido primeiro seria o numero 11.

.

## **Decremento**

O decremento e a mesma coisa do incremento, so que invés de somar ele vai subtrair.

.

## **Operador de inversão de sinais**

let n=10

let x=-n

console.log(x)

Podemos converter os sinais com o codigo acima, o resultado imprimido sera -10.

.

## **concatenação**

let n1 = 10;

let n2 = 20;

console.log(n1 + n2); /\*Mostra o valor da soma entre n1 e n2 \*/

console.log(n1 + "" + n2); /\* Faz uma concatenção inves de somar \*/

no exemplo abaixo serão somadas as 2 variaveis ee sera mostrada o resultado da soma.

console.log(n1 + n2); /\*Mostra o valor da soma entre n1 e n2 \*/

Neste exemplo abaixo invés de fazer uma soma eremos na verdade uma concatenação oque sera mostrada vai ser “10 vs 20”.

console.log(n1 + "vs" + n2); /\* Faz uma concatenção inves de somar \*/

.

# Aprendendo sobre operador ternário - Curso de Javascript Moderno - Aula 10

E uma forma de reduzir ou sumplificar uma operação com uma condicional, podemos utilizar operadores relacionais e operadors lógicos para mostrar esse resultado de uma forma melhor do que true e false.

## **Condicional**

let num=11

let res=num%2

if(res==0){

    console.log(`par`)

}else{

    console.log("impar")

}

Aqui temos uma operação em condicional, temos a variável num com 11 e a variável res com num%2 que eo resto da divisão entre num e 2. Entao o resto de 11 dividido por 2 vai ser igual a 1.

Se(resto da divisão entre num e 2 for igual a 0){

mostrarNaTela(`Par`)

}senao{

Mostrar(`Impar`)

}

Entao resumindo se 0 o resultado do calculo em res for igual a 0 devera ser mostrando par, senão(else) ou caso não seja igual a 0 será mostrado impar.

.

## **Operador ternário**

// Teste logico ? se verdadeiro(1) : se false(0)

Abaixo temos um exemplo de operador ternário

rest=(2+2==4 ? "true(1)" : "false(0)")

console.log(rest)

Aqui ja no inicio do codigo vemos que rest vai pegar o resultado de todo esse calculo

E o calculo eo seguinte se 2+2 for igual a 4 sera mostrado true(0) senão se for falso sera mostrado false(0).

O resultado foi true(0) pois e uma verdade.

Devemos lembrar que

True e igual a 1 e false e igual a 0.

## **Outro exemplo**

Inves de colocar todo aquele codigo condicional nos usamos apenas

let num=10

res=(num%2 ? "Par" : "Impar")

console.log(res)

//0 = False

//1 = True

Nesse codigo temos uma variável chamada num com o numero 10, mas abaixo aqui já simplificamos um pouco as coisas

res=(num%2 ? "Par" : "Impar")

Deveremos lembrar pela explicação abaixo que o resto de num%2 acima e igual a 0

Esse calculo acima e quase a mesma coisa do codigo condicional que já fizemos la em cima mas logo você vera que teremos um problema, continue lendo esse codigo, pegue o resultado de num%2 e se for igual a 1 sera imprimido par se se for igual a 0 sera imprimido Impar, mas o porque disso? Se você reparar bem estamos com valores invertidos pois o restante de zero deve dar par e não impar, isso e porque

Devemos lembrar primeiramente que false(falso) e igual a 0, e true(verdadeiro) e igual a 1.

//0 = False

//1 = True

Essa e uma regra básica da programação

? "Par" : "Impar"

Mas o problema e que o resultado deu 0, e 0 e igual a false e quando e false sera mostrado a segunda opção que neste caso e “impar” mas nos queremos que quando sera 0 que seja mostrado par e não impar, entao resumindo esse calculo esta apresentando o resultado de forma errada pois 10%2 o restante e 0 mas e igual a false, mostrando assim a segunda opção que e impar.

Faremos seguinte entao

let num=10

res=(!(num%2) ? "Par" : "Impar")

console.log(res)

Pronto resolvemos colocando uma negação entao se o resultado de num%2 for igual a 0 que e false sera invertido para true, pois estamos usando uma negação no codigo, e com isso resolvemos

.

Podemos ter a seguinte questão vamos supor que eu esteja recebendo dados de um banco de dados, e criamos uma variável onde ela rec ebe o status de um cliente onde se ele for ativo ele envia a letra a e se for inativo o banco de dados envia a letra i

let st="a"

res=(st == "a" ? "Ativo" : "Inativo")

console.log(res)

Aqui fizemos o seguinte se o status do cliente for igual a “a” de ativo sera mostrado “ativo” mas caso seja diferente de “a” que provavelmente sera i pois so temos essas 2 opções no banco de dados sera mostrado “inativo”.

.

# Operador Typeof, retornando o tipo da variável - Curso de Javascript Moderno - Aula 11

Com o Typeof conseguimos saber com exatidão o tipo de variavel que estamos lidando.

Abaixo temos os seguintes tipos de dados

let v1=10//Numero

let v2="10"//String

let v3=v1===v2//boolean

let v4={nome:"Bruno"}//Objeto

**v1** = Sao todos os números

**v2** = São as string que são formatos de textos são mostrados oque realmente estão escritos

let v2="10 + 10"//String

se fosse um numero por exemplo o resultado imprimido no console.log() seria 20, mas como e uma string ele mostra exatamente oque esta escrito entao o resultado e 10 + 10.

**V3** = Esse e ul tipo Booleano que eo de comparação para compararmos se e igual um maior que o outro < e etc.

**V4** = Esse tipo se chama objeto, mais pra frente veremos sobre ele.

.

.

## **typeof**

Mas não precisamos decorar o tipo de dados pois podemos ver qual e ele com um codigo e as vezes precisamos de saber qual o tipo de um certo dado no codigo para podermos por exemplo converter para outro tipo de dado

No codigo abaixo usamos o typeof para descobrir o tipo de dados da variável v2, e descobrimos que o tipo dela e string

let v2="10 + 10"//String

console.log(typeof(v2))

e será imprimido o tipo de dado, não precisa ser so no console.log podemos por exemplo jogar o resultado em uma variável

res=typeof(v2)

Ai dentro de res estará escrito string pois foi jogado dentro de res o tipo de dado de v2.

.

# Operador Spread, aprendendo sobre o espalhador em Javascript - Curso de Javascript Moderno - Aula 12

Ele quebra um array, e vai devolver elemento a elemento.

Vai simplificar muito quando se trata de array

Nos podemos concatenar objetos dentro do codigo

const jogador1={nome:"Bruno",energia:100,vidas:3, magia:150};

const jogador2={nome:"Bruce",energia:100,vidas:5, velcidade:80};

const jogador3={...jogador1,...jogador2}

console.log(jogador3)

Se por exemplo temos o objeto jogador1 com seus dados, e o objeto jogador2 com seus dados, depois nos pegamos e usamos spread no jogador 3 criando tipo uma junçao do jogador1 com o jogador2 formando assim o jogador 3, o resultado sera

{ nome: 'Bruce', energia: 100, vidas: 5, magia: 150, velcidade: 80 }

Como os dois já tinham nome: foi usado somente o do ultimo, energia os dois tem entao foi mantido e vida foi utilizada a do ultimo colocado, mas agora magia e velocidade e uma característica de cada um e foi pego e colocado no jogador 3.

.

Podemos usar o spread da seguinte maneira,

const soma=(*v1*,*v2*,*v3*)=>{

    return *v1*+*v2*+*v3*

}

let valores=[1, 5, 4]

console.log(soma(valores))

Aqui temos um array function que faz a soma do valor v1, v2, v3 e la embaixo no console.log nos chamamos essa function, so que os dados nos pegamos dentro da array

let valores=[1, 5, 4]

mas quando vamos executar vemos que da o seguinte erro

1,5,4undefinedundefined

Ele além de não somar ficou todo bagunçado, oque acontece e que na hora da function pegar os números dentro da array let valores e jogar la dentro pra somar, ela vai toda desorganizada e pois isso usamos o spread, ele vai espalhar os números la dentro, cada um dentro do local certo.

const soma=(*v1*,*v2*,*v3*)=>{

    return *v1*+*v2*+*v3*

}

let valores=[1, 5, 4]

console.log(soma(...valores))

.

Neste novo exemplo nos começaremos criando divs dentro do nosso JavaScript

const objs = document.getElementsByTagName("div")

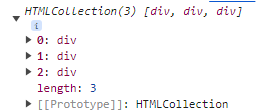
logo depois nos pegamos a tagname div e jogamos dentro de objs.

O document que tem em alguns codigos servem para o DOM

Se terntarmos

console.log(objs)

vai dar error pois isso não funciona no cscode ou editor de codigo pois isso seria pro lado do servidor mas devemos ver isso do lado do cliente, entao podemos colocar la no console do navegador que poderemos ver as divs e suas respectivas numerações



Faremos spread em nosso obj entao da seguinte maneira

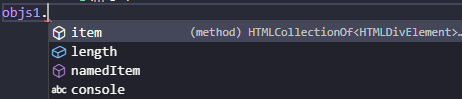
const objs1 = document.getElementsByTagName("div")

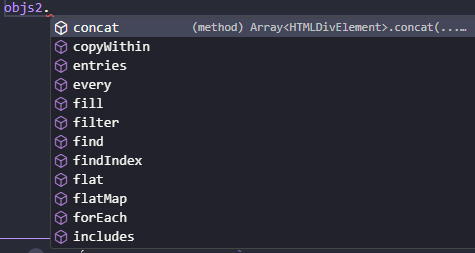
const objs2 = [...document.getElementsByTagName("div")]

Como podemos ver o obj1 eo que fizemos anteriormente, agora o obj2 nos colocamos dentro de uma array e fizemos o spread, mas oque isso nos beneficiaria?

Isso da uma lista mais longa de opções pois o obj1 que e oque chamaos de HTM collection que e do lado do cliente que e do html (lado do navegador), nos poderemos usar opções so do html, mas agora se fizermos ele como um array e colocarmos um spread nos podemos utilizar ele tambem como uma array e com as funcionalidades de um array.

Podemos comparar isso logo abaixo, o tanto de opções que conseguimos utilizando o spread e fazendo ele como uma array.





.

Agora vamos supor que eu queira percorres pelos elementos do obj1 e retornar o conteúdo usando forEach

const objs1 = document.getElementsByTagName("div")

objs1.forEach(*element* => {

    console.log(*element*)

});



Vai dar erro sem o sprad

.

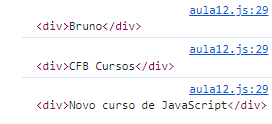
Mas agora vamos tentar fazer a mesma coisa com o obj2

const objs2 = [...document.getElementsByTagName("div")]

objs2.forEach(*element* => {

    console.log(*element*)

});



Agora deu certo nos retornou(mostrou) os elementos html.

.

Agora nos podemos fazer uma mlehor manipulação ate do HTMl

Podemos ate mesmo mudar o texto das 3 divs que eu peguei e coloquei no objs2

objs2.forEach(*element* => {

*element*.innerHTML="curso"

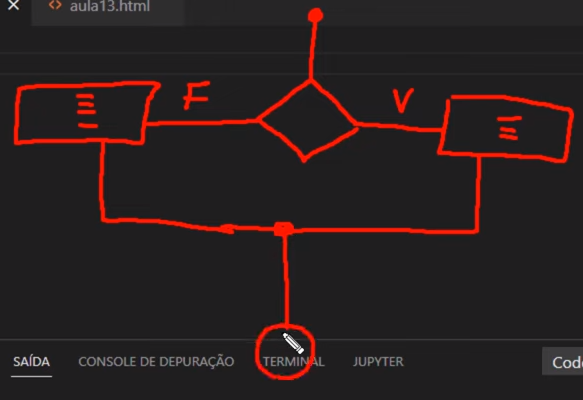
});



.

# Comando condicional IF e IF ELSE em Javascript - Curso de Javascript Moderno - Aula 13

A condicional ela vai englobar uma expressão logica retornardo true ou false, mas podemos escolher, entao por exemplo se for retornado true eu posso escolher que execute um certo bloco de códigos, mas caso seja false ele continua o codigo normal.



Temos o diagrama acima para utilizar como exemplo, aqui la no inicio do diagrama nos temos o codigo normal, mas quando chega naquele trapézio podemos dizer que aparece o teste condicional, entao se o teste logico for verdadeiro elevai executar o codigo de nossa escolha representando no diagrama como o v do lado direito, mas se for false ele executara o codigo da esquerda e após isso ele termina de executar o codigo normal.

Agora faremos um exemplo de codigo mas da nossa maneira

Se (5>4){

Bloco de codigo 1

}senão{

Bloco de codigo 2

}

Isso e como se fosse o codigo, se 5 for maior que 4 ele executara o bloco de codigo 1 mas se não for ele executara o bloco de codigo 2, o resultado sera o bloco de codigo 1 pois 5 e maior que 4.

Em codigo JavaScriopt ficara assim

if(5>4){

    console.log(`Bloco de codigo 1`);

}else{

    console.log(`Bloco de codigo 2`)

}

Podemos colocar a quantidade que quisermos de codigo entre as chaves.

.

Podemos tambem aumentar esse teste condicional colocando mais opções]

Veja o exemplo abaixo

let num= 10

Se (num <=3){

Escreva(“num e maior que 3”)

}senaoSe (num >6){

Escreva(“num e maior que 6”)

}senaoSe (num >7){

Escreva(“num e maior que 7”)

}senão{

Escreva("num e maior que todos os numeros”)

}

.

let num= 10

if(num<=3){

    console.log('num e maior que 3')

}else if(num>6){

    console.log("num e maior que 6")

}else if(num>7 && num!=100){

    console.log("Num e maior que 7")

}else{

    console.log('nukm e maior que todos os numeros')

}

Aqui temos diversos teste condicionais, o teste que der true sera executado mas caso nenhum de certo o ultimo escrito else e executado.

PODEMOS COLOCAR TESTE CONDICIONAIS DENTRO DE TESTES CONDICIONAIS

.

# Comando Switch Case em Javascript - Curso de Javascript Moderno - Aula 14

O Switch availia uma expressão e de acordo com, o resultado dessa expressão, ele compara com os cases que ele ta programado para mostrar o resultado, e se por caso algum coincidir com os cases que ele tem o case e executado.



Esse e nosso diagrama

* nesta parte e colocado a nossa expressao que queremos ai logo vamos para o case 1, ai se por acaso o primeiro case for igual a expressão que colocamos nesse prisma, o codigo dentro dele ser executado, mas caso não seja igual ele vai pro segundo case e dai por diante ate achar um case que tenha aver com a expressão colocada no topo, se não for encontrado nenhum sera executado aquele final do codigo escrito def. que significa (defalt) ele serve para caso não tenhe nenhum case que atende os requisitos la da expressão esse defalt e executado, e depois continua o resto do codigo pra baixo e esse e nosso switch case.

.

Temos um exemplo de codigo abaixo

let colocacao = 1

switch(colocacao){

    case 1:

        console.log("Primeiro Lugar")

        break

    case 2:

        console.log("Segundo Lugar")

        break

    case 3:

        console.log("Terceiro Lugar")

        break

    default:

        console.log("Não subiu ao pódio")

        break

}

Aqui nos temos uma variável chamada colocação com o lugar que o jogador conseguiu ficar queo eo primeiro, podemos ver tambem que logo abaixo temos cases, são 3 cases e cada um com uma característica, oque esser codigo faz eo seguinte caso colocação seja igual a 1 ele mostrada “Primeiro Lugar”, mas se caso colocação for igual a 2 o resultado sera “Segundo Lugar”, mas caso a colocação for igual a 3 sera mostrado “Terceiro Lugar” , mas se por um acaso colocação não for igual a nenhum desse casas sera executado o defalt mostrando assim “Não subiu ao pódio”, devemos lembrar que esses brakes dentro de cada case e do defalt e essencial para o funcionamento do codico e os requisitos dos cases não prescisam ser somente números, como no exemplo abaixo:

case “Legal”:

        console.log("Voce e um cara legal")

        break

Podemos colocar quantos cases quisermos

Podemos fazer op seguinte tambem

 case 3: case 4: case 5:

        console.log("Terceiro Lugar")

        break

Varios cases para um so bloco de codigo.

.

# Loop FOR em Javascript - Curso de Javascript Moderno - Aula 15

Uma estrutura em loop e uma estrutura que vai9 repetir um bloco de comandos ou um comando so, e sempre que prescisarmos repetir a execução de um conjunto de instruções nos utilizamos um loop, cada repetição que o loop fazer nos estamos fazendo uma interação.

Os loops nos podemos classificar entre definidos e indefinidos, loops que vamos usar quando sabemos a quantidade de precisamos de repetir ou os indefinidos quando não sabemos quantas vezes ao certo que vamos repetir.

**Loops Definidios:** For.

**Loops indefinidos:** While, Do While.

### **A sinaxe do for ea seguinte**

For(inicialização; condição; contador){

}

Oque estiver dentro das chaves sera repetido.

### **Temos o seguinte exemplo:**

for(let i=0; i < 10; i++ ){

    console.log(i)

}

Aqui nos inicializamos criando uma variável e colocamos 0 dentro dela, depois criamos uma condição apara a petição que fara que o loop continue enquanto i for menor que 10, e colocamos i++, onde sempre que a estrutura de loop repetir sera incrementado 1 á variável **i.**

E o codigo que sera executado dentro desse loop eo console.log(i), isso fara com que mostre o valor da variável a cada repetição, entao resumindo enquanto i for menor que 10 a variável i sera incrementada e mostrara o novo valor da variável i no console, uma outra explicação e que enquanto a condição for verdadeira(true) ele continuara o loop, entao o resultado do codigo sera uma contagem de 0 ate 9.



.]

# Loops FOR IN e FOR OF em Javascript - Curso de Javascript Moderno - Aula 16

Os números em javaScript sempre são contando começando do zero, entao em qualquer contagem não sera 1, 2 3, 4 mas sim 0, 1, 2, 3, 4)

Primeiro devemos entender o .lenght.

## **.length**

Faz com que não mostre o valor mas sim a quantidade, entao se temos a seguinte lista array

let num=[10,20,30,40,50];

console.log(num.lenght)

temos 5 elementos dentro dela, entao com o num.length invés de mostrar os números dentro da array, sera mostrado a quantidade de números que temos dentro dessa array, e no exemplo acima nos temos 5 numeros entao o valor retornado sera 5.

nosso exemplo temos o seguinte codigo:

let num=[10,20,30,40,50];

for(let i=0; i<num.length; i++){

    console.log[i]

}

Esse codigo e o seguinte primeiro temos uma array, com alguns números [10, 20, 30, 40, 50], e logo embaixo temos uma estrutura de repetição FOR, onde na repetição colocamos uma inicialização que i recebe 0, ai em nossa condição enquanto i for menor que num.length o loop sera repetido, e no contador nos colocamos um incremento que sempre quando tiver essa repetição haverá incremento no i, e sera mostrado o valor de i no console.log.

Entao o codigo sera executado 5 vezes pois o valor de num.length e 5 porque tem 5 numeros dentro dessa array, mas se quisermos mostrar o valor dentro da array um por um temos uma técnica com o loop for

let num=[10,20,30,40,50];

for(let i=0; i<num.length; i++){

    console.log(num[i])

}

Já notou alguma diferença?

Sim dentro do copnsole.log nos agora colocamos num[i], você se lembra que para acessarmos um numero de uma array precisamos de pegar o nome dessa array e colocar entre colchetes o indicie da posição do elemento que estamos procurando?

Entao olhando o array acima para acessarmos o numero 30 nos devemos utilizar

    console.log(num[2])

que é a posição dele no índice.

Entao agora e muito simples, como sempre que repetimos o loop e incrementado 1 a variavel i e a variável i esta dentro de num(i), entao cada hora sera mostrado uma posição dentro do índice de num, fazendo assim que seja mostrado todos números dentro da array num.

### **Resultado**

Info: Start process (14:27:59)

10

20

30

40

50

Info: End process (14:27:59)

.

## **For In**

Este codigo faz a mesma coisa do anterior

let num=[10,20,30,40,50];

for(n in num){

    console.log(num[n])

}

Esse for in faz o seguinte, ele primeiro pega e ve qual coleção que queremos percorrer, neste caso a coleção ea num que e nosso array, ai ele percorre toda essa coleção e coloca os índices dentro de n, que e como se fosse uma variável dentro desse loop colocando os índices que tem dessa coleção, entao sempre que esse loop percorre a array e encontra um elemento la dentro ele repete denovo, ate acabar os elementos, ai no final dentro de n temos a quantidade de elementos que existem dentro dessa array, e enquanto tudo isso acontece, a array em conjunto com seu índice esta sendo mostrado dentro do console.log mostrando assim todos os números.

Entao se eu quiser pegar as posições da array dentro de n nos usamos o for in.

Podemos fazer isso tambem no html

.

## **For of**

for(n of num){

    console.log(n)

}

O for of e bem parecido com o for in, ele percorre toda a coleção e já conta automaticamente a quantidade de elementos que tem dentro da coleção e de acordo com quantidade de elementos será a mesma quantidade que ele será repetido, entao ele vai percorrendo, so que invés dele percorrer e colocar dentro de nos índices da coleção, ele já coloca os elementos direito, entao ele não vai colocar os índices, mas sim o conteúdo do índice até acabar todos.

Entao se eu quiser pegar o conteudo dos índices da array e colocar dentro de n nos usamos for of

.

## **For of e For in no navegador (DOM)**

const objs =document.getElementsByTagName("div");

for(o of objs){

    console.log(o.innerHTML="curso")

}

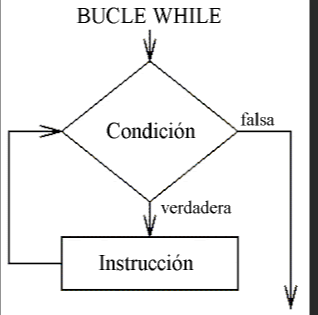
Podemos pegar as divs por exemplo de uma pagina html e colocar a palavra “curso” de acordo com a quantidade de divs.

.

# Loop While em Javascript - Curso de Javascript Moderno - Aula 17

O loop while e um loop indefinidio (Quando não temos certeza da quantidade de vezes que queremos executar esse loop.

## **Diagrama (While)**



Enquanto a condição for verdadeira o loop continuara, mas apartir do momento que a condição se tornar falsa, o loop vai parar e sair para continuar outros códigos.

.

**While = enquanto**

let n=0

while(n<10){

    console.log(n)

}

Neste exemplo acima nos temos um loop infinito, primeiro nos criamos uma variável chamada n com o valor zero, depois começamos a estrutura de repetição while, e enquanto n for menor que 10, sera mostrado no console a variável n, mas sera infinito pois n sempre sera menor que 10 porque a variável n neste caso não muda, a variável n sempre sera 0 e sempre sera menor que 10.

Mas agora se incrementarmos algum valor a variável n ai nos podemos deixala finita invés de infinita.

while(n<10){

    n++

    console.log(n)

}

Agora neste novo exemplo nos estamos incrementando 1 a variável n, fazendo assim uma contagem no nosso console de 1 ate 10.

Mas so lembrando que esses códigos são exemplos, podemos usar a estrutura de repetição para outra diversas meneiras, podemos mudar a codição e etc.

.

Por exemplo podemos criar um programa para calcular o fatorial de 5

let n=5

let fat = 1

while(n>=1){

    console.log(fat)

    fat\*=n

    n--

}

.

# WHILE e DO WHILE em Javascript, entenda a diferença - Curso de Javascript Moderno - Aula 18

Ambos vao executar quando uma expressão for verdadeira, ambos são loops que nos vamos usar em situações indefinidas, que e quando nos não sabemos quantas vezes que vamos repetir o loop.

Essa seria a estrutura em português do do-while

Faça{

Codigo aqui

}enquanto(condição)

Entao diferente do while comum, o do-while e executado ao menos uma vez, e se vai fazer um loop ou não depende se a expressão for verdadeira ou falsa, mas agora o while se a condição for falsa não sera executada nenhuma vez e não sera executado.

Entao o Do-while primeiro executa o codigo e pode repetir se for verdadeiro, mas se for falsa não repete, e o while comum se for verdadeiro ele fara o loop normalmente mas se for falsa não e executado nenhuma vez, e simplesmente ignorada.

let n=1

do{

    console.log("teste")

}while(n=10)

Neste exemplo do do-while o console.log primeiro e executado e depois vai para a condição para ver se o codigo vai ser executado mais uma vez, mas não vai porque a variável n não e igual a 10 entao segue o codigo kmas já mostrando a palavra teste uma vez pois já foi executado.

.

# Entenda as declarações BREAK e CONTINUE em Javascript - Curso de Javascript Moderno - Aula 19

## ***break***

O break e uma interrupção na execução, entao encontrou o comando break, ele para o comando loop e continua a execução normal do codigo, ele para totalmente o loop.

let n=0;

let max=1000;

while(n<max){

    console.log("CFB Cursos - " + n);

    if(n>10){

        break;

    }

    n++

}

console.log("Fim do programa")

por exemplo no codigo acima nos temos a estrutura de repetição que enquanto n for menor que max ele continua sendo executado e max tem o valor 1000, entao o coimando seria executado 1000 vezes, más temos uma estrutura condicional com if onde, quando n for maior que 10, ele tem um break, e quando cai no break o loop e parado de vez e termina a execução desse loop, fazendo assim que a contagem seja somente ate 11.

.

## **Continue**

O continue quando ele e executado dentro de um loop ele para somente aquela interação e pula para a próxima interação ele so não vai executar a interação atual e vai pula para a próxima, mas continua dentro do loop.

.

# Aprendendo sobre FUNÇÕES em Javascript [#P1](https://www.youtube.com/hashtag/p1) - Curso de Javascript Moderno - Aula 20

Uma função e um bloco de comando que podemos executar em um momento que prescisarmos ou escolhermos, vamos uopr que estamos executando todo o codigo normalmente, mas ai criamos uma função no meio desse codigo, ele n ao sera executado pois ele so e executado quando e chamado, entao ele vai pular a função e vao continuar o codigo, mas quando a função for chamada ai sim ela sera executada, temos diversar maneiras para executa-la, como um click, mover mouse, load e etc...

function nomeDaFunction(){

}

Essa e uma estrutura de uma function onde escrevemos function para inicializar, e depois escolhemos um nome pra function e depois do nonme sempre devemos colocar parênteses, esses parênteses sao obrigatórios ter, mas não quer dizer que precisamos preencher ou usar mas os parentes prescisam ficar la mesmo que vazios, esses parênteses recebem um parâmetro, ai depois abrimos efechamos chaves e dentro das chaves que colocamos nossos códigos, entao se tiver rodando um codigo normal ai o codiugo chegar em uma function, não vai acontecer nada ele simplesmente vai pular a function, mas quando chamamos ela da seguinte maneira

nomeDaFunction()

Ai quando o codigo chegar nessa parte, a function la em cima vai ser executada, ai todo o codigo dentro da function vai ser executada.

Olhe o exemplo abaixo

function soma2\_10(){

    let n1=2

    let n2=10

    let soma= n1+n2

    console.log(soma)

}

for(let i=0; i<10;i++){

    soma2\_10()

}

Pode parecer um pouco complexo mas na verdade e bem simples, temos uma function e uma estrutura de repetição for que já aprendemos, se não tiver entendendo volte para as anotações de estrutura de repetição for, ai na estrutura de repetição enquanto a variável i for menor que 10 o loop sera repetido, e sera chamado a function

    soma2\_10()

entao como esta chamando a function, enquanto i for menor que 10 a function soma2\_10 sera chamada

.

.

# FUNÇÕES com retorno em Javascript [#P2](https://www.youtube.com/hashtag/p2) - Curso de Javascript Moderno - Aula 21

Podemos usar o return da seguinte maneira, olha que interessante

O return não faça nada mais além de retornar algo, uma informação entao veremos um exemplo abaixo

function canal(){

    return "CFB Cursos"

}

Nessa function canal(), nos temos um return e esse return vai retornar a string “CFB Cursos”, nos poderíamos colocar dentro do console.log para poder aparecer no terminal, mas nos faremos dessa vez fazendo um retorno, esse codigo vai retornar a string “CFB Cursos, entao quando pegarmos essa function e mostrar no console sera mostrado oque foi retornando, ou melhor ele vai retornar oque esta com return e sera mostrado.

console.log(canal())

Aqui nos pegamos o console.log e dentro dele colocamoa function, oque sera mostrado e obvio, sera mostrado o retorno que e “CFB Cursos”, isso porque CFB Cursos recebeu um return

A função **canal()** retorna a string "CFB Cursos", e quando você chama **console.log(canal())**, o valor retornado pela função é exibido no console.

É uma forma útil de encapsular um pedaço de código que realiza uma operação específica e retorna um resultado, permitindo que você reutilize esse código em diferentes partes do seu programa.

Parte superior do formulário

.

# FUNÇÕES parametrizadas em Javascript [#P3](https://www.youtube.com/hashtag/p3) - Curso de Javascript Moderno - Aula 22

Uma função parametrizada significa que queremos entrar com valores para dentro da função, nos entramos com os parâmetros dentro dos parênteses, e eles são executado dentro da function.

**Exemplo:**

function soma(*p1*){

    console.log(*p1*)

}

O valor dentro do parâmetro (p1) e colocado dentro da function para ser usado, e neste exemplo nos o utilizamos para ser mostrado no console.log(), mas antes precisamos dar um valor a esse parâmetro e mostraremos como.

Para colocar um valor no parâmetro nos podemos chamando a própria function

function soma(*p1*){

    console.log(*p1*)

}

soma("Cfb Cursos - JavaScript")

neste exemplo criamos a function com o parâmetro p1 e dentro da function o console.log vai imprimir o conteúdo de p1, ai embaixo da function temos uma chamada, que chama a function para ser executada, mas assim que e chamada entre os parênteses temos o valor que sera atribuído ao nosso parâmetro p1, entao ao mesmo tempo que chamou a function essa ultima linha de codigo tambem atribuiu um valor ao parâmetro p1.

E toda vez que chamarmos a function atribuindo um novo valor ao parâmetro, ele sera executado e mostrado todas as vezes.

function soma(*p1*){

    console.log(*p1*)

}

soma("Cfb Cursos - JavaScript")

soma("E um otimo curso")

soma("John CHK")

## **Resultado**

Info: Start process (23:58:29)

Cfb Cursos - JavaScript

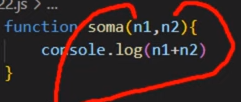
E um otimo curso

John CHK

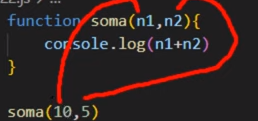
Info: End process (23:58:30)

.

Podemos colocar 2 parametros tambem



Neste exemplo nos criamos uma function chamada soma com 2 parametros: n1 e n2 e dentro dessa function temos uma operação de soma, entre a soma de n1 e n2, mas ate o momento não temos valores para nossos parâmetros.



Entao agora nesta nova imagem nos chamamos a function e colocamos 2 valores para os parâmetros, entao colocamos o numero 10 para o n1 e o numero 5 para o n2.

O resultado da soma sera 15

.

E podemos passar quantos parâmetros quisermos

## **Mas e se caso não passarmos nenhum valor para um parâmetro?**

Caso não passemos nenhum valor ao parâmetro mas mesmo assim utilizamos ele, sera mostrado Nan pois não tem valor, mas para resolver isso podemos passar valores padrões para eles, entao caso um parâmetro não tenha sido atribuído nenhum valor podemos colocar valores padrões

### **Exemplo**

function soma(*n1*=0, *n2*=0){

    console.log(*n1*+*n2*)

}

soma(2)

Neste exemplo acima passamos o valor apenas de um parâmetro, mas colocamos valores padrões a eles, entao como colocamos o valor 10 a n1 e o n2 já esta com 0 por padrão, o resultado da soma sera 2.

.

Podemos utilizar return tambem

function soma(*n1*=0, *n2*=0){

    return *n1*+*n2*

}

console.log(soma(2))

primeiro vai ser chamado a function levando o numero 2 ao parâmetro n2 depois a soma sera feita e retornara o resultado, ai como chamamos o console.log na chamada da function, sera mostrado o valor em nosso console, e o resultado continua sendo 2.

.

Outra situação

function soma(*n1*=0, *n2*=0){

    return *n1*+*n2*

}

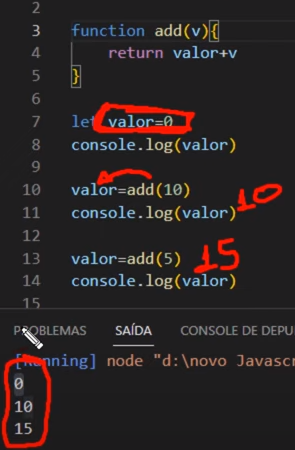
let resultado\_soma =soma(5,5)

console.log(resultado\_soma)

Essa e a mesma coisa mas a function será chamada vai executar a function fazendo o calculo, ai o return vai retornar o valor e jogara dentro da variável resultado\_soma, e depois vai ser imprimido no console.log().

.

Outra situação



.

# Parâmetros REST em funções Javascript [#P4](https://www.youtube.com/hashtag/p4) - Curso de Javascript Moderno - Aula 23

Quando criamos as function com o no exemplo abaixo]

function soma(*n1*=0,*n2*=0){

    return(*n1*+*n2*);

}

console.log(soma(10,5))

Nos temos dois parâmetros, mas somente dois parâmetros, e cso queira colocar mais eu tenho que especificar os parâmetros dentro da function, entao o REST veio para resolver isso, poderemos ter quantos parâmetros nos quisermos sem precisar especificar cada um.

function soma(...*valores*){

    return *valores*.length;

}

console.log(soma(10,5));

Aqui neste exemplo acima nos criamos um rest chamado valores, o rest e criado com 3 pontos na frente do nome, logo ali dentro da function criei um return que vai retornar o legth da REST chamada valores, entao ele vai retornar a quantidade de parâmetros que contem a REST valores, pois uma REST armazena apenas parâmetros, mas la em baixo no console.log chamamos a function chamada soma e colocamos dois valores para os parâmetros, e como colocamos dois valores para o RESt logo o REST criou dois parâmetros para armazenar esses dados, e entao o resultado daquela function sera 2, pois oque foi retornado no console.log e o length de valores que e 2.

function soma(...*valores*){

    return *valores*.length;

}

console.log(soma(10,5,6,7,2));

Neste novo exemp0lo acima temos 5 valores para os parâmetros, entao o REST criara 5 parametros e armazenara cada um desses valores, e o resultado que sera retornado vai ser 5 pois são 5 valores que teremos.

.

Essa function abaixo calcula todos os valores que tem no parâmetro valores

function soma(...*valores*){

    let tam = *valores*.length;

    let res=0;

    for(let i=0; i<tam;i++){

        res+=*valores*[i]

    }

}

console.log(soma(10,5,6,7,2));

Primeiro criei uma variável chamada tam que armazena o numero de valores que tem a REST valores, entao depois cria uma variável chamada res que tem o valor padrão de 0 mas dentro do looping for ele recebe cada valor dos parametros e soma tudo, e no looping for enquanto i ser menor que tam, que é a quantidade de parametros dentro de valores o for continua rodando ate tam seja do tamanho de i e logicamente tenha passado por todos os parametros de valores.

.

Podemos faze-lo usando for of tambem que já aprendemos a usar antes

function soma(...*valores*){

    let res = 0

    for (let n of *valores*) {

        res+=n

    }

    return res

}

console.log(soma(10,20,30))

Agora utilizando tambem o for of podemos somar os valores de valores.

.

# Funções Anônimas em Javascript [#P1](https://www.youtube.com/hashtag/p1) - Curso de Javascript Moderno - Aula 24

São funções que não tem nomes associados ao seu corpo ou seu conteúdo, são funções que são chamadas em tempo de execução, ela so e criada no momento da execução, ela não fica pronta na memoria antes de ser chamada.

Uma função comum ela e criada e fica guardada esperando somente que seja chamada em alguma parte do codigo.

.

## **Exemplo de função anonima**

const f=function(*v1*,*v2*){

    return *v1*+*v2*;

}

console.log(f(10,5))

A diferença neste exemplo e que na função anonima não precisamos usar um nome pausar um nome para chamar a function anonima.

.

## **Construtor**

Toda vez que falarmos sobre construtor devemos nos lembrar do operador new, que pode mudar a construção da nossa aplicação, ele serve para operações mais simples.

const f=**new** *Function*("v1","v2","return v1+v2")

console.log(f(10,5))

Dois detalhes, primeiro colocamos o new antes do function e o function deve iniciar com F maiúsculo, neste exemplo acima estamos usando uma function construtor dentro de uma function anonima.

Prescisamos passar os argumento, e o corpo da função, neste exemplo os argumentos são o v1, v2, v3 e o corpo da função sera o return v1+v2.

Oque manda e sempre o ultimo parâmetro da função construtor, pois ele sera o corpo da função, resumindo sera oque a função vai fazer com esses parametros.

.

# Arrow Function em Javascript. Aprenda o que é e como usar! - Curso de Javascript Moderno - Aula 25

Funções lambidas (arrow function)

E um tipo de declaração de função na anonima,

const soma=function(*v1*, *v2*){return *v1*+*v2*} //Função comun

const soma=(*v1*,*v2*)=>{return *v1*+*v2*} // arrow function

Neste exemplo acima tems um exemplo de function comum e uma arrow function, os dois códigos trazem o mesmo resultado so que com técnicas diferentes, podem ver que tiramos o function que estava escrito, e tambem apontamos com uma seta => para onde a function seria executada.

Se for uma função que vai entrar so um parâmetro não precisamos colocar parênteses, ficando assim

const nome = *n*=>{return *n*}

Na verdade no padrão array function nem prescisariamos do return, e se não tiver return na array function nao precisamos nem das chaves

const add=*n*=>*n*+10

ficando desta maneira, mas isso continua sendo uma function.

Mas e claro que não usaremos chaves nemparentes somente se for um codigo bem simples, quee prescise de so uma linha, senão teremos de usar toda a sintaxe dele.

const add=*n*=>*n*+10

console.log(add(10))

Podemos ver acima, primeiro nos criamos ali a array function sem usar parênteses nem chaves, a array function faz uma soma de n+10, mas ai quando colocamos o parâmetro ali dentro do console.log, foi adiciona 10 a variável n ficando assim o resultado se tornou 20.

## **Sintaxe completa do array function**

let variavel = (*parametros*)=>{

    /\* CODIGO \*/

}

.

# Funções dentro de funções em Javascript. Funções aninhadas - Curso de Javascript Moderno - Aula 26

Temos aqui uma função dentro de outra função

const soma=(...*valores*)=>{

    const somar=*val*=>{

        let res=0

        for(v of *val*){

            res+=v

            return res

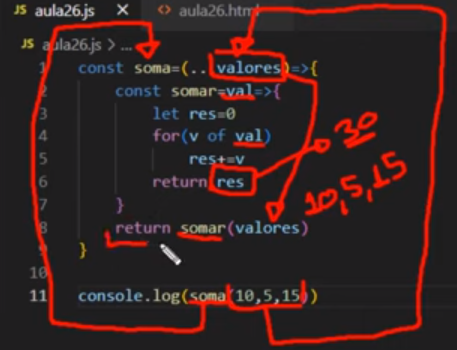
        }

    }

    return somar(*valores*)

}

console.log(soma(10,5,15))



.

# Funções Geradoras em Javascript. Você precisa aprender! - Curso de Javascript Moderno - Aula 27

Uma função geradora tem seu retorno adiado ate que nos prescisemos desse retorno, ou melhor essa função e pausada mantendo os valores dentro dela ate que prescisemos dela novamente, entao a execução nunca e parada mas sim pausada, entao sempre que executarmos essa função e quisermos voltar mais tarde e executala novamente o valor que estava la dentro antes ainda estrara la, e ela são o pilar da programação assíncrona, quando uma função normal e chamada quem fica com o controle dessa função e a própria função ela tem o seu controle ate o final que ela retorna alguma coisa, mas as funções geradoras a própria função pode ter esse controle quando quiser retornar alguma coisa, resumindo não precisa de todo o processamento para retonar algo no final, entao podemos ir retornando coisas ao longo da função, e toda vez que chamarmos a função ela pode retornar algo diferente.

Para criar uma função geradora precisamos usar a palavra function\* com um asterisco no codigo, yield retorna o conteúdo no codigo.

function\* cores(){

    yield 'Vermelho'

    yield 'verde'

    yield 'Azul'

}

const itc = cores()

console.log(itc.next().value)

Então no codigo acima, temos uma função geradora que deverão retornar Vermelho, verde e azul, la em baixo em

const itc = cores()

console.log(itc)

já fizemos uma primeira chamada para a function que no console.log aparecera assim

Object [Generator] {}

Nos mostra que o conteúdo da variável itc que é a function cores() é um object [Generator] {}, mas agora para retornarmos os valores dessa function prescisaremos de um outro chamado

console.log(itc.next().value) // Vermelho

Esse chamado com o .next() que e um método da função geradora, fara uma execução em nosso codigo retornando vermelho em nosso console. Agora se chamarmos a função mais duas vezes retornara verde e azul

console.log(itc.next().value) // Vermelho

console.log(itc.next().value) // Verde

console.log(itc.next().value) // Azul

mas e se agora chamarmos a função novamente o valor sera undefined pois não tem mais valores para retornar..

Novo exemplo

function\* perguntas(){

    const nome=yield 'Qual seu nome?'

    const esporte=yield 'Qual seu esporte favorito?'

    return `Seu nome é ${nome}, seu esporte favorito é ${esporte}`

}

const itp=perguntas()

console.log(itp.next().value)

console.log(itp.next('Bruno').value)

console.log(itp.next('Natação').value)

neste exemplo nos criamos uma função de retorno chamada perguntas(), e dentro dela criamos duas variáveis que receberão valores que serao inseridos nos chamados, na primeira execução, resumindo o resultado sera

Qual seu nome?

Qual seu esporte favorito?

Seu nome é Bruno, seu esporte favorito é Natação

.

Outro exemplo

function\* contador(){

    let i=0

    while(true){

        yield ++i

    }

}

const itc = contador()

console.log(itc.next().value) // 1

console.log(itc.next().value) // 2

console.log(itc.next().value) // 3

console.log(itc.next().value) // 4

Neste exemplo, aquela estrutura de repetição se não fosse uma função geradora seria infinita, mas nesse caso eele retornara mais um cada vez que chamarmos essa função pois essa função mantem o valor anterior sempre, entao antes era 1 e depois a execução foi pausada, a próxima vez que chamarmos manetera o 1 e colocara mais 1, entao a função foi executada 4 vezes ai.

function\* contador(){

    let i=0

    while(true){

        yield i++

        if(i>5){

            break

        }

    }

}

const itc = contador()

for(let c of itc){

    console.log(c) // 1 ate 5

}

Aqui terá uma contagem ate 5, pois na verdade isso e uma execução in finita mas como dentro do while determinamos que apartir do momento que a variável i valer 5 a execução sera terminada usando o break.

.

# Aprendendo sobre o método MAP em Javascript - Curso de Javascript Moderno - Aula 28

Função map tem a funcionalidade de percorrer arrays (coleções), basicamente vamos utilizar o map quando precisarmos trabalhar elemento por elemento de uma coleção, diferente dos loops como for in, for of o map ele vai iterar toda a coleção, usaremos quando precisarmos operar por completo os elementos de uma coleção.

Entao quer dizer que podemos percorrer arrays e passar os argumentos por parametros como no exemplo abaixo

const cursos = ['HTML','CSS','JavaScript','PHP','REACT']

cursos.map((*element*,*indice*)=>{

    console.log("Curso: "+ *element* + " - Posição do curso: " + *indice*)

});

Aqui temos um array chamada cursos, com diversos cursos de programação, entao embaixo criamos um método .map utilizando o cursos ficando cursos.map, ou quer dizer que vamos percorrer o cursos utilizando o .map, agora fica mais simples ainda pois criamos 2 parametros dentro da função .map, e dentro de uma função .map sempre que criamos os parametros o primeiro parâmetro passa os elementos que tem dentro daquela array e o segundo parâmetro passa o índices dela ou se preferir melhor podemos chamar de posições, e como se funcionesse como um loop tambem so que ele percorre toda array, entao ele vai pegar o item da array vai jogar dentro do parâmetro element e depois pegar o índice dela e jogar dentro do parâmetro chamado índice que criamos que e nosso segundo e ultimo parâmetro que criamos ali dentro, lembrando que os parametros podem se chamar de qualquer maneira, eu que escolhi que se chamassem element e índice, e criamos uma array function que retorna no console.log() um texto contendo os elementos e os índices respectivos de cada um, você pode ver no codigo, e como o .map percorre todo a array entao serão retornados todos os elementos.

O resultado quando executado deste codigo é

Curso: HTML - Posição do curso: 0

Curso: CSS - Posição do curso: 1

Curso: JavaScript - Posição do curso: 2

Curso: PHP - Posição do curso: 3

Curso: REACT - Posição do curso: 4

Podemois ver que ele percorreui a coleção inteira e retornou no console.log.

O map já e próprio para percorrer toda a coleção além de ser mai clen, mais otimizado utiliza-lo nestas situações.

.

Olha que interessante tambem esse codigo abaixo

const cursos = ['HTML','CSS','JavaScript','PHP','REACT'];

let c=cursos.map((*element*,*indice*)=>{

    return "<div>"+*element*+"</div>"

 });

console.log(c)

Isso retornara em nosso console

[

  '<div>HTML</div>',

  '<div>CSS</div>',

  '<div>JavaScript</div>',

  '<div>PHP</div>',

  '<div>REACT</div>'

]

So que não veremos isso por agora, isso sera mais pra frente mas e so para termos uma previa.

.

Abaixo temos um codigo em HTML em uma pagina HTML.

Nosso objetivo sera percorrer esses elementos e usar o map para operar de alguma maneira.

    <div *id*="c1">HTML</div>

    <div *id*="c2">CSS</div>

    <div *id*="c3">JavaScript</div>

    <div *id*="c4">PHP</div>

    <div *id*="c5">React</div>

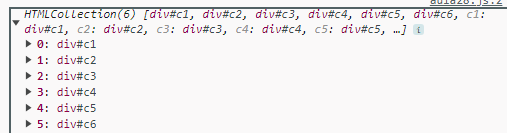
    <div *id*="c6">MySQL</div>

Primeiro para acessa-los em nosso codigo JavaScript nos devemos pegar todas as div da pagina ou somente a que queremos, mas como vamos pegar todas div da pagina entao usamos os seguintes codigo

const el=document.getElementsByTagName('div');

console.log(el)

Aqui nos acessamos todas TAG div da pagina e colocamos dentro da variável el e mostramos no console.log(), mas lembrando que e como e codigo HTML, ele não sera mostrado no terminal do seu editor de codigo mas sim no browser, entao entramos no console do browse e poderemos ver o retorno do console.log que criamos.



Oque foi retornado no console do browse foi este.

Mas agora se colocarmos o .map

const el=document.getElementsByTagName('div');

el.map((*e*,*i*)=>{

    console.log(*e*)

});

Normalmente esse codigo retornaria todos os elementos para o console, mas vai dar erro pois esse el não e uma coleção iterável (não sei oque significa se vira), mas podemos deixa-la iterável utilizando o spread.

const el=document.getElementsByTagName('div');

el2=[...el]

el2.map((*e*,*i*)=>{

    console.log(*e*)

});

Ficando desta maneira que esta em cima, entao nos criamos outra variável fizemos el virar uma array e utilizamos o spread, so que pra que criar outra variável?

let el=document.getElementsByTagName('div');

el=[...el]

el.map((*e*,*i*)=>{

    console.log(*e*)

});

Podemos simplesmente transformar a própria el em um spread e utiliza-la, agora o map posso trabalhar da forma que eu quiser, e já ta funcionando no console do browse.

Agora vamos alterar o conteúdo do elemento

Devemos nos lembrar que para alteramos um texto de algum elemento HTML usamos o .innerHTML=”texto”, entao neste caso como vamos alterar todas as divs de uma vez e colocaremos o mesmo texto nos utilizamos o seguinte codigo;

let el=document.getElementsByTagName('div');

el=[...el]

el.map((*e*,*i*)=>{

*e*.innerHTML ="CFBCursos"

});

Primeiro nos pegamos os elementos Div da pagina html, criamos um array com ele utsando spread, depois colocamos na função .map seguindo de uma array function que tem como parâmetro o e que são os elementos e o i que é o índice, e como o parâmetro e são os elementos de el, logo serão as nossas div e mudaremos com o innerHTML, logo todos elementos div da pagina foram modificadas e tem o texto CFBCursos.]

.

Outra maneira

const elements = document.getElementsByTagName('div');

const val = *Array*.prototype.map.call(elements, ({*innerHTML*})=>*innerHTML*)

Resumindo esse codigo seleciona todos os os elementos div do documento e extrai o conteúdo HTMl interno de cada uma delas e armazena esses conteúdo dentro de uma nova array chamada val.

O Array. transforma todo as divs em uma coleção array, já o .map nos já vimos la em cima e o .call nos permite usar o.map para transforma a coleção de div em um array de seus conteúdos internos.

, ({*innerHTML*})=>*innerHTML*)

Essa parte do codigo pega o innerHTML que e o conteúdo de nosso elementos (no caso as DIVs),

.

Outro exemplo

const converterInt=(*e*)=>parseInt(*e*)

let num=['1', '2', '3', '4', '5'].map(converterInt)

console.log(num)

Aqui nos temos um codigo que converte todos as strings de uma array para números, criamos uma função com o parâmetro e e com uma array function nos fazemos um parseInt no parâmetro e que faz os números virarem inteiros, logo abaixo temos uma coleção de array que e recebida por uma variável chamada num e colocamos o .map para percorrer todos elementos desta array e passa-los pela function converterInt que converte os números para inteiro, e o num recebe tudo isso, então logo ele recebera o resultado disso tudo que são todos os elementos daquela array convertidos para numero, e no final imprimimos no console.log().

.

So para lembra-los que o método .map sempre pega três parametros de uma array, que e o elemento, o índice (posição) e o próprio array.

.

# Usando o operador THIS dentro de funções em Javascript - Curso de Javascript Moderno - Aula 29

O This funciona da seguinte forma, olhemos o exemp0lo abaixo

function aluno(*nome*,*nota*){

    let nome=*nome*

}

Neste exemplo nos queremos que uma variável chamada nome receba o parâmetro que tambem se chama nome, mas os dois tem o mesmo nome, entao o JavaScript não entedera oque queremos que seja feito, oque podemos fazer e criar uma variável com o nome diferente para receber o parâmetro nome.

function aluno(*nome*,*nota*){

    let vnome=*nome*

}

Mas desta maneira não vai nos satisfazer pois nos queremos de qualquer maneira que o nome da variável seja o mesmo nome, entao não podem ser diferentes, nesse caso usaremos o operado this.

function aluno(*nome*,*nota*){

*this*.nome=*nome*

}

Esse this sera como uma variável declarada para a função aluno que receber o o valor do parâmetro nome, criando assim uma distinção entre ás duas.

function aluno(*nome*,*nota*){

*this*.nome=*nome*

*this*.nota=*nota*

    console.log(`O nome é ${*nome*} e sua nota foi de ${*nota*} pontos.`)

}

aluno("Bruno", 100)

Neste exemplo o parâmetro nome recebe bruno, e o parâmetro nota recebe 100, logo em seguida são criadas tipo umas variáveis com o mesmo nome dos parametros e receber os valores dos parametros, e logo em seguida sera mostrado o nome dessas novas varivaies no console.log().

.

Outro exemplo

function aluno(*nome*,*nota*){

*this*.nome=*nome*

*this*.nota=*nota*

*this*.dados\_anonimo=function(){

        setTimeout(function(){

            console.log(*this*.nome)

            console.log(*this*.nota)

        },2000)

    }

}

const aluno1 = **new** aluno("Bruno", 100)

aluno1.dados\_anonimo()

temos esse outro exemplo sobre o this, temos o codigo antigo que já mostramos más agora criamos outra function anonima dentro da nossa function anterior, onde o nome da function anonima e dados\_anonimo, e dentro dela nos temos um SetTimeOut que serve para criar tipo um temporizador para a reprodução do codigo e colocamos um tempo de 2 segundos(2000ms), Entao este SetTimeOut vai demorar 2 segundos para fazer a function funcionar e quando a function funcionar podemos ver que dentro dela nos temos dois console.log(), um com this.nome e o outro com this.nota, más temos uma peculiaridade, pois quando esse codigo for rodado o resultado no console sera dois undefined, muitos acharam que iria mostrar o valor de nome e de nota no console mas oque vocês não sabem e que mesmo que essa função esteja dentro da outra onde foi criado o this.nome e this.nota e mesmo que elas tenham recebido valores, o setTimeOut bloqueia elas, tipo sombreia o this.nome, e o this.nota, entao quando chamamos no console.log o this.nome e o this.nota eles la dentro já não existem mais, mas agora se esse this.nome e this.nome for declarado dentro do setTimeOut e receber os valores ai vai funcionar

function aluno(*nome*,*nota*){

*this*.nome=*nome*

*this*.nota=*nota*

*this*.dados\_anonimo=function(){

        setTimeout(function(){

*this*.nome="x"

*this*.nota=5

            console.log(*this*.nome)

            console.log(*this*.nota)

        },2000)

    }

}

const aluno1 = **new** aluno("Bruno", 100)

aluno1.dados\_anonimo()

Oque sera mostrando no console sera o x e o 5, mas não os outros valores atribuídos anteriormente, mas tem uma maneira de resolver isso com arrow function mostrando outra vantagem dele.

function aluno(*nome*,*nota*){

*this*.nome=*nome*

*this*.nota=*nota*

*this*.dados\_arrow=function(){

        setTimeout(()=>{

            console.log(*this*.nome)

            console.log(*this*.nota)

        },2000)

    }

}

const aluno1 = **new** aluno("Bruno", 100)

aluno1.dados\_arrow()

Agora que tiramos a função anonima e colocamos a arrow function ira funcionar normalmente, oque sera mostrado no console são os valores atribuídos anteriormente fora dessa função do setTimeOut, serão mostrados o nome que e Bruno e a nota que é 100 pois o arrow function olha a função pai, não a função que ele ta dentro, ele olha a função pai de todas que e a aluno.

.