**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 ate 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 varias 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 agente 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 da 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 nos 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 sera utilizado la dentro tambem.”

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

.

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

Podemos mudar algo que esta 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 sera imprimido vai ser o numero 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, ai pegamos a variável res e jogamos dentro dela o resultado da soma de num1 + num2 que sera 15.

.

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

num1=5

num2=10

console.log(num1-num2)

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

.

## **Ordem de precedencia**

num1=5

num2=10

console.log(num1-num2)\*2

nesse caso pensando na ordem de precedência primeiro sera num1-num2 por causa dos parênteses, e depois o resultado sera 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 sera mostrado no console.log sera 1 pois e 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 e num1 e colocamos ++ sera adicionado um numero, entao como fizemos 2 vezes o valor final da variável num1 sera 12.

.

## **decremento**

num1=10

num1--

num1--

Aqui nesse caso nossa variável num1 tem o valor 10, mas o incremento sempre retira um numero dessa variável entao quando colocamos o nome da variável que e num1 e colocamos -- sera retirado um numero, entao como fizemos 2 vezes o valor final da variável num1 sera 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 nod podemos escolher quanto queremos incrementar.

num1 = 10;

num1+= 5;

neste exemplo sera incrementao 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 sera 13 pois o valor de num1 era 10 ai foi incrementado o valor de num2 ficando assim o resultado final 13.

.

## **Decremento PRO**

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

num1 = 10;

num1-= 5;

neste exemplo sera 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 sera 7 pois o valor de num1 era 10 ai 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 atribuido a ele é maior que num2 que tem o numero 5 atribuido a ele, o resultado sera 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 sera verdadeiro(true) senão se não for o resultado sera falso(false).

O resultado sera verdadeiro(true) pois num1 e igual a num3 mesmo que não seja mior, 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 atribuido é menor que num2 que tem o numero 5 atribuido a ele, o resultado sera 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 sera verdadeiro(true) senão se não for o resultado sera falso(false).

O resultado sera 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 sera verdadeiro (true) pois num1 e igual a num3.

.

### **= Atribuição**

Atribuição nos 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) numero 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 sera verdadeiro(true) pois num1 e igual a num3, mas ai nos adicionamos o sinal NOT a ele.

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

nos adicionamos e colocamos os valores entre colchetes, o sinal de NOT(!( )) ele nega, entao se o valor deu verdadeiro(true), sera inverso para false, e se o valor der false sera 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 e uma string, um numero entre outras coisas, entao além de comparar os valores ele compara o tipo de dados e tambem o lugar na memoria.

Temos tambem o 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 memoria 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 e uma verdade mas o NOT ele simplesmente pega a comparação e inverte o resultado, em tao 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.

.

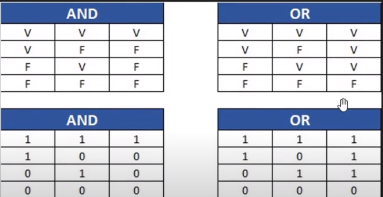
# Operadores Lógicos em Javascript, aprendendo a usar! - Curso de Javascript Moderno - Aula 07

**&&** -> 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

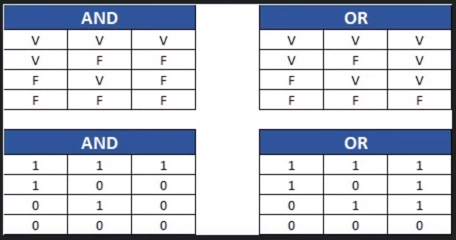
let b = false;

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

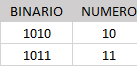
No nosso exemplo aqui os três primeiros se repetem entao o resultado sera 0 e o ultimo não se repete entao o resultado final e 1.

.

## **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")

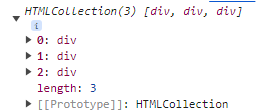
logo depois nos pegamos a tagname div e jogamos dentro de objs.

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

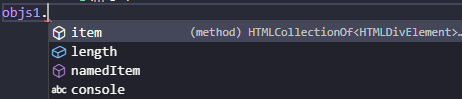
const objs1 = document.getElementsByTagName("div")

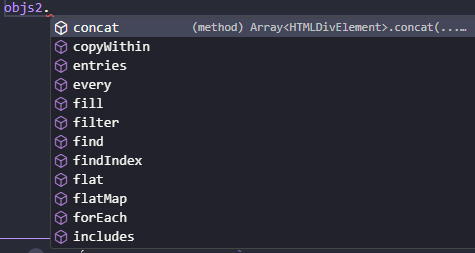
const objs2 = [...document.getElementsByTagName("div")]

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

.

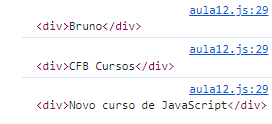
Mas agora vamos tentar fazer a mesma coisa com o obj2

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"

});



.