**Fundamentos do JavaScript: tipos, variáveis e funções**

Fundamentos – Porque é importante vermos? Para resolvermos bugs o quanto antes

JavaScript é conhecida como a “linguagem da web”, embora tenha se desenvolvido no front-end, atualmente é mais utilizado no back-end, ou server side (lado do servidor) das aplicações web.

Usaremos durante o curso o NodeJS

**Tipos de dados**

Tem três tipos principais que são os primitivos:

* Números (number) – salva os números
* Strings (string)– salva os caracteres
* Booleanos (boolean)– para armazenar verdadeiros ou falsos
* Null – é um tipo especial, pois pode ser traduzido como “ausência de valor” e pode ser atribuído como valor de uma variável (por exemplo, se queremos verificar se uma entrada já possui um valor, avaliamos com o null)
* Undefined – também representa “ausência de valor”, porém de uma outra forma, já que o null se refere a um valor atribuído a uma variável que existe e foi iniciada, e undefined se refere ao valor de uma variável que não foi inicializada.

Criando uma constante: const nomedaconstante = valor;

Podemos usar os operadores + - \* / para as operações básicas da matemática

Math.PI = retorna o valor de pi

Math.round(valorparaarredondar) – arredonda números decimais

.toFixed(númerodecasas) – consegue controlar o número de casas decimais após a vírgula

toLocaleString() – converte um número para uma string

toLocaleString(‘pt-BR’, {style: ‘currency’, currency: ‘BRL’} – serve para transformar um número na moeda real brasileira, que tem esses argumentos:

* Style: o estilo do formato a ser utilizado, que pode usar:
  + Decimal – para representar números simples
  + Currency – colocar valores para moeda
  + Percent – para representar percentuais

Math.ceil() – retorna o maior número inteiro que é maior que o número passado – exemplo: Math.ceil(11.123)=12

Math.floor() – retorna o menor número inteiro que é menor que o número passado, exemplo: Math.floor(11.789)=11

**Ordenar números no JavaScript**

Let numeros = [0,1,2]

numeros.sort(); - essa função serve para ordernar números em uma lista array

Porém, pode ser que não ocorra da ordem esperada, e deve ser utilizada a seguinte solução

O sort recebe opcionalmente uma função de comparação que, dados dois valores, deve devolver um número inteiro:

* Se for 0 indica que são iguais
* Se for -1 indica que o primeiro valor é menor
* Se for 1 indica que o segundo valor é menor

function comparaNumeros(a,b) { if (a == b) return 0; if (a < b) return -1; if (a > b) return 1; }

lista.sort(comparaNumeros); // [1, 5, 8, 9, 10, 12, 15]

e podemos enxugar da seguinte forma:

lista.sort((a, b) => a - b); // [1, 5, 8, 9, 10, 12, 15]

**Como executar no terminal do VSCode um arquivo JS usando node**

node arquivo.js

**Ponto flutuante (números decimais)**

Podemos colocar só usando o ponto mesmo para representar essa casa decimal, de forma direta: 3.3, e se usarmos o 0,5 por exemplo, usamos apenas: .5

**Not a number (não é um número)**

Isso acontece caso uma operação matemática não tenha um número envolvido, por exemplo se usarmos uma string e multiplicamos, irá retornar esse erro

**Tipo string (tipos textos)**

Serve para guardar tipos textos, tudo o que não for número ou booleano, podemos guardar caracteres, pontuação, tudo.

Const texto1 = “Mensagem aqui”; - para criar uma string, abrimos as aspas duplas, e podemos fazer em aspas simples também, logo, para string não há diferença para o uso de aspas simples ou duplas na hora de criar uma string. Mas para fazermos uma citação, devemos abrir as aspas simples e fazer a citação em aspas duplas, e o mesmo serve da forma contrária, da seguinte forma

const citação = ‘Reirysson disse: “oi”’;

**Template string ou template literal**

Template strings são as que permitem expressões embutidas, podemos usar string multi-linhas e interpolação de string com elas, as quais são envolvidas pelos acentos gravar (` `), e podem possuir placeholders, que são indicados por um cifrão seguido de chaves (${expression}), essas, bem como o texto em volta delas são passados a uma função, uma função padrão apenas concatena as partes em uma string única, se existir uma epressão precedendo a template string (função tag exemplo) a template string é definida como “tagged template string”, no caso, a expressão tag é chamada pela template string processada, que você pode manipular antes de produzir o resultado. Para isso, temos o seguinte exemplo:

Const nome = “Reirysson”;

Const profissão = “Programador”;

Console.log(`Meu nome é ${nome} e eu sou um ${profissão}`);

**Concatenação – juntar duas strings (+)**

Podemos usar o sinal de adição para juntar duas ou mais strings, por exemplo:

Const citação = “Meu nome é: “;

Const nome = “Reirysson”;

Console.log(citação+nome); - “Meu nome é: Reiryson”;

**toLowerCase() – converter todos os caracteres da string para letra minúscula**

exemplo:

const cidade = “belo horizonte”;

const input = “Belo Horizonte”;

const inputMinusculo – input.toLowerCase();

console.log(cidade === inputMinusculo); // true

**lenght – para saber quantos caracteres uma string possui**

const senha = “minhasenha123”;

console.log(senha.length) // 13 caracteres

**const – para variáveis que não se alteram**

**Tipo booleano**

Não é um número ou texto, e ele só tem dois valores, que é true ou false, é um dos mais usados para questão de comparação, em, lações if, for, while.

= - atribui um valor a uma variável

== - compara só os valores que estão dentro

=== - compara o valor e o tipo

Podemos fazer de forma direta usando no console.log por exemplo e comparar valores, strings.

**Case-sensitive – diferença de letras maiúsculas e minúsculas**

Temos algumas convenções (boas práticas e padronização) para se usar:

* camelCase – para palavras que inicia com letra minúscula e a primeira letra de cada palavra seguida é com letra maiúscula
* snake\_case – para onde todas as palavras são minúsculas
* kebab-case – para vai ter variáveis e funções
* PascalCase – para casos em que todas as palavras começam com letra maiúscula
* **Nunca colocar espaço e iniciar uma declaração com número**

**Variáveis**

Temos três tipos, que são os seguintes:

* Var – o fato de a var poder ser declarada e depois ser substituída, acaba deixando a mesma solta, não tem uma regra especifica e acaba resultando em certos eventos que não sejam previstos. É uma variável global, logo funciona em qualquer lugar que declare a variável – **é muita propensa a bugs**
* Let – É parecido com o var, pois consegue armazenar o valor, manipular, só que tem uma diferença de escopo, e o mesmo só funciona dentro do local onde foi declarado. Exemplo:
  + If(true) {

Let a = 10;

}

Console.log(a);

Dessa forma, irá aparecer um erro, pois o let a só funciona dentro do if, já dessa forma:

If(true) {

Let a = 10;

Console.log(a);

}

Irá funcionar, pois o let a está declarado dentro do mesmo laço de repetição

* Const - tem o mesmo escopo do let, pois funciona só dentro do local onde foi declarado, só que tem uma particularidade diferente do let, logo a constante não pode ser alterada, uma vez definida, será o valor que foi definido como constante, isso para números inteiros simples, mas já para objetos conseguimos acessar a propriedade e conseguir fazer a alteração, logo uma constante não pode ser reatribuída mas não quer dizer que ela não pode ser modificada se ela for do tipo que tenha propriedades, em que essas podem ser modificada. Sempre que a variável constante, temos que atribuir um valor para iniciar

**Booleanos – Truthy e falsy**

Não são booleanos, mas podem comportar-se como se fosse, ou seja, como se fossem falsos e verdadeiros.

Alguns utilizam 0 e 1 para determinar esses valores booleanos

0 = false , strings vazias também são consideras como false “”

1 = true

Console.log(typeof variável) – vai retornar o tipo dessa variável, em que retorna uma string indicando o tipo de um operando

**Conversão de tipos**

Temos no JavaScript as seguintes conversões:

* Conversão implícita – podemos converter um tipo de dado em outros, por exemplo, um número para uma string, uma string para um número. Exemplo:

Const numero = 456;

Const numeroString = “456”;

Console.log(numero === numeroString);

Dessa forma, será retornada que é falsa essa comparação, pois um número não é igual a uma string, mas para fazer a conversão do número para string, podemos fazer da seguinte forma:

Console.log(numero == numeroString);

Logo, assim será retornado o valor de true.

* Conversão explícita

Mas podemos fazer da seguinte forma, que é a mais recomendada>

Number() – pra transformar uma string para um número (se caso a string tiver uma caractere, ele vai retornar NaN, pois não tem o valor correto)

String() – pra transformar um número para uma string – também tem o toString() é uma outra forma para fazer essa conversão, que é mais parecida com outras linguagens de programação

**Palavras para não colocar em declarões:**

arguments

as

async

await

break

case

catch

class

const

continue

debugger

default

delete

do

else

eval

export

extends

false

finally

for

from

function

get

if

import

in

instanceof

let

of

new

null

return

set

static

super

switch

target

this

throw

true

try

typeof

var

void

while

with

yield

**e para o javascript**

enum

implements

interface

package

private

protected

public

**JavaScript e Node.js**

JavaScript (Padronizada pela ECMA - ECMAScript) – linguagem que chamamos de tipagem dinâmica, ou seja, não precisamos determinar qual é o tipo da variável, por exemplo, podemos criar let minhaVar = 567; e depois atribuir o seguinte valor, minhaVar = “texto”; logo, não precisamos determinar a variável da seguinte forma let minhaVar numer = 567; pois ele verifica qual o tipo da variável e atribui.

Também pode ser dito como untyped que não precisa declarar os tipos das variáveis.

É uma linguagem multiparadigma, ou seja, podemos resolver o problema de várias formas. Porque chama de JS – e ela não é igual ao java

ES6 / ES2015 – é a versão do JS que saiu em 2015, pois teve algumas versões anuais

Linguagem interpretada – escreve o código e vai ser executada o que está escrito (não precisa ser compilada)

Linguagem compilada – passa primeiro para um compilador e transforma para a linguagem de máquina, e vai ser executado no computador

NodeJS – é o JS, não é uma linguagem, e sim um interpretador (o navegador também é um interpretador de JS), se usarmos o Node para interpretar o JS, estaremos focados em usar o back-end. O node é uma ferramenta que foi criada depois do JS para usar ele fora do navegador.

**Erros e stacktrace**

SyntaxError (erro de sintaxe) – foi escrita alguma coisa errada – na maior parte dos casos ocorre quando há erros no programa e o JS não consegue executá-lo

ReferenceError (erro de referência) – quando o código tenta acessar algo que não existe, como uma variável que não foi definida

rangeError – quando ó código recebe um dado do tipo certo, porém não dentro do formato aceitável. Por exemplo, um processamento que só pode ser feito com número inteiros maiores ou igual a zero, mas recebe -1, irá apresentar o RangeError

TypeError – indica que o código esperava receber um dado de um determinado tipo, tal qual uma string de texto, mas recebeu outro, como um número, booleano ou null

Stacktrace – o caminho que o código executou e as ferramentas que foram usadas para o mesmo

Documentação de erros node.js <https://nodejs.org/api/errors.html#errors_errors>

**Console.api**

Console é uma ferramenta que tem tanto no node como nos navegadores, que serve para colocar frases, dados para fora da aplicação caso o usuário preciso

Console.log(); – é um registro

Console.error – usa principalmente quando encontrar algum erro, podemos colocar alguma frase, alguma coisa (tratamento de erro). Para usar usamos: console.error(‘Deu erro’) – isso para o node

Console.table() – para visualizar de forma mais organizada informações tabulares

Console.time() e console.timeEnd() – para temporizar período que uma operação de código leva para ser iniciada e concluída

Console.trace() – para exibir a stacktrace de todos os pontos (ou seja, os arquivos chamados) por onde o código executado passou durante a execução

Documentação do console node.js <https://nodejs.org/api/console.html>

**Operadores de comparação**

É uma característica do JS

= - atribuição de valor em uma variável

== - faz a comparação e ele faz a conversão implícita antes de fazer a comparação, ou seja, só compara o valor (comparação implícita)

=== - sempre compara os valores, sem fazer a compara antes, ou seja, compara o tipo de dado e o valor, logo, um número e uma string aqui não são iguais, já com dois iguais, eles vão ser iguais (comparação explícita), as boas práticas usam eles.

|| - operador “ou”

&& - operador “e”

!= e !== - operadores “não igual” e “estritamente não igual” – da mesma forma que o == e o ===

Documentação sobre operadores <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Expressions_and_operators>

**Operador ternário**

Retorna verdadeiro e falso, faz basicamente um if, uma comparação, vai descobrir se uma operação é verdadeira ou falsa.

Exemplo:

Const idadeMinima = 18;

Const idadeCliente = 16;

Console.log(idadeCliente >= idadeMinima ? “Cerveja” : “suco”);

Logo, é quase igual a um if, em que pegamos a idadeCliente e vemos se ela é maior igual a idadeMinima e perguntamos com o ? se for true vai sair cerveja e se for false, vai sair suco

Mas para a boa prática com mais comparações, fazemos com o IF, e caso seja curto, como esse exemplo, fazemos com o operador ternário, por conta que é três operadores

**Template literal**

Para tipos strings (templates strings) – escreve a string normal, e além de concatenar os valores, colocamos o `${string}`

**Funções**

Precisamos de funções para que possamos fechar um bloco de código e a gente pode dizer o que quer que funcione dentro desse pedaço de código. Em que as funções podem receber informações de outros locais para que as mesmas sejam executadas. Como declarar uma função:

function nomedafunção(parâmetro) {

o que vai fazer na função

}

Como chamar a função:

Nomedafunção(valor para a informação);

Fazendo uma função sem um parâmetro:

function soma(){

const resultado = 2+2;

}

Console.log(soma());

- se colocarmos apenas assim, não teremos resultado no console.log, em que o mesmo retornará undefined, e para resolvermos isso, podemos usar o return, que é o que a gente quer que volte como return. E que podemos fazer assim:

function soma(){

return 2+2;

}

Console.log(soma());

Em que agora irá apresentar o resultado. E com isso, podemos até usar duas funções de forma conjunta, da seguinte forma: imprimeTexto(soma()); - em que o resultado irá ser o return da função soma

O JS executa uma linha por linha

**Formas de escrever uma função no JS**

A forma anterior, é chamada de **declaração de função**, que é a forma mais clássica

**Expressão de função**, é uma outra forma de fazer em JS, em que fazemos da seguinte forma:

const soma = function(num1, num2) { return num1 + num2 }

A diferença dessa forma, é que não temos o nome da função, em que é opcional. E é uma forma mais curta de fazer uma função (anônima).

Conseguimos fazer uma função em linhas diferentes, ou seja, criar a função em baixo e chamar ela em uma linha de cima, antes de ter declarado a mesma, isso para uma função de declaração, já nas expressões de função, não conseguimos fazer isso, pois, quando fazer a declaração com const, não podemos usar uma variável sem antes de ter iniciada a mesma. E vale lembrar que funções e var são “listadas” no topo, e sendo a diferença principal chamada de hoisting

**Arrow function,** é uma função que usa =>, veio junto com a ES6, deixa tudo em uma linha, para usarmos mais de uma linha precisamos abrir as {} e não pode ser nomeada. Essa é uma forma rápida de fazer uma função, para o código ficar mais curto, e se comporta como hoisting também. que fazemos da seguinte forma:

const variável = parâmetros => instruções da função;

const apresentarArrow = nome => `Meu nome é ${nome}`;

const soma = (num1, num2) => num1 + num2;

**Funções prontas de matemáticas (math)**

* Math.round() – faz o arredondamento de um número de ponto flutuante para o inteiro mais próximo: Math.round(4.3) – retorna 4 / Math.round(2.5) – retorna 5, se terminar com 5 o número, ele arredonda para cima
* Math.ceil() – faz o arredondamento para o valor mais alto, também conhecido como teto (ceil): Math.ceil(5.2) – retorna 6
* Math.floor() – faz o arredondamento para o valor mais baixo, também conhecimento como piso (floor): Math.floor(5.2) – retorna 5
* Math.trunc() – desconsidera os números decimais, o que é conhecido como truncamento: Math.trunc(4.3) – retorna 4 / Math.trunc(4.8) – retorna 4
* Math.pow() – faz a exponenciação de 2 números, quando for simples, como ao quadrado (2) ou ao cubo (). Recomenda-se usar a multiplicação por ser mais rápido: Math.pow(4,2) – retorna 4^2 = 16
* Math.sqrt() – retorna a raiz quadrada de um número: Math.sqrt(64) – retorna 8
* Math.min() – retorna o menor valor entre os argumentos: Math.min(0, 150, 30, 20, -8, -200) – retorna -200
* Math.max() – retorna o maior valor entre os argumentos: Math.max(0, 150, 30, 20, -8, -200) – retorna 150
* Math.random() – retorna um valor randômico (random) (um número aleatório) entre 0 e 1, então não teremos um valor esperado para receber: Math.random() – retorna 0.7456916270759015

Documentação do Math: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Math#description>

Guida de git e github: <https://www.youtube.com/watch?v=BAmvmaKQklQ&t=9s>

**Parâmetros e argumentos**

Na prática se referem ao mesmo tipo de dado, algumas documentações se referem a parâmetros no momento em que a função é definida (no caso, numero1, numero2, etc), e argumentos como os dados que utilizamos para executar a função (ou seja, 30, 45, etc)

Parâmetros de função – servem para a função receber informações para executar de forma correta essa função, por exemplo, se termos uma função onde vai haver uma soma de números, devemos ter como parâmetros o recebimento de números. Para fazermos uma função com parâmetros, fazemos da seguinte forma:

function soma(n1, n2) {

return n1 + n2;

}

console.log(soma(2,2)); - logo terremos como resultado 4

Da mesma forma, podemos colocar duas funções para fazer, pode exemplo: console.log(multiplica(soma(4, 5), soma(3, 3))); - logo teremos a multiplicação da soma de 4+5 e 3+3, ou seja, 9 \* 6 = 54. E se não passarmos um dos parâmetros da função, teremos o erro NaN

**Definindo um parâmetro**

Podemos definir um parâmetro para que não ocorra erros, da seguinte forma:

Function multiplica(numero1 = 1, numero2 = 1) {

Return numero1 \* numero2;

}

Logo, podemos passar só um parâmetro e mesmo assim não dará um erro.

**OBS.: OS PARÂMETROS SÓ VALEM PARA DENTRO DA FUNÇÃO, OU SEJA, PODEMOS TER DUAS FUNÇÕES COM AS MESMAS VARIÁVEIS, MAS ELAS SÓ IRÃO SE COMPORTAR DENTRO DA FUNÇÃO, E NÃO TERÁ CONFLITO ENTRE AS MESMAS**

**Ordem dos parâmetros** – nos parâmetros, tem uma ordem de preenchimento das variáveis, ou seja, se colocamos que a primeira variável qualquer tipo de variável, ele vai preenche, mesmo que seja como outra. Exemplo:

function nomeIdade(nome, idade) {

return `Meu nome é: ${nome} e minha idade é: ${idade}

}

console.log(nomeIdade(40, Thiago)); - dessa forma a frase ficará a seguinte: Meu nome é 40 e minha idade é: Thiago, ou seja, ele só preencheu os parâmetros nas variáveis, seguindo a ordem que tem presente na função.

**Fundamentos do JavaScript: Arrays**

É mais focado em back end

**Array (lista)** – é um tipo de estrutura de dados, que é uma coleção ordenada de valores, ou seja, cada valor chama de elemento, e pertence a sua própria posição.

Definindo uma lista: const notas = [10, 6.5, 8, 7.5]

Como acessar uma posição da lista: nomedalista[0] – sempre começa do zero uma posição de uma lista, em que cada posição é um índice e cada número é um elemento.

Encontrando o tamanho de uma lista: nomedalista.length

**Adicionar elementos a uma lista (pelo método do array)**

lista.push(7); – vai adicionar ao último elemento esse valor – exemplo [10, 6, 8] – [10, 6, 8, 7] – se não adicionarmos um valor em push, ele vai exibir o erro NaN

**Deletando elementos a uma lista**

lista.pop(); - vai retirar o último elemento da lista – exemplo [10, 7, 8] vai ficar – [10, 7]

**Alguns outros métodos:**

* **concat()** – Junta dois arrays, colocando o array passado como argumento, logo depois do primeiro, ou seja, é uma concatenação. E não altera o array no qual foi chamado, então precisamos salvar esse resultado em um novo array
* **filter() –** Retorna uma lista contando todos os elementos que passaram em um teste, ou seja, uma função escrita por nós. E não altera o array no qual foi chamado, então precisamos salvar esse resultado em um novo array
* **find() –** Funciona de forma parecida com o filter, porém retorna apenas o primeiro valor que satisfazer o teste, podendo ser uma string ou um número
* findIndex()
  + Funciona igual o find(), mas retorna o índice em vez do elemento, possibilitando usá-lo em outras partes do código.
* indexOf()
  + Localiza e retorna o índice referente à primeira ocorrência de determinado valor em um array. Caso não exista nenhuma ocorrência do valor, retorna -1.
* lastIndexOf()
  + Funciona da mesma forma que o indexOf(), porém retorna o índice referente à *última* ocorrência de um valor em um array, varrendo o array de trás para frente.
* forEach()
  + Executa uma função em cada elemento do array de forma individual.
  + Não altera o array original e nem retorna um valor, deixando esse trabalho a cargo da função escolhida.
* pop()
  + Retira o último elemento do array.
  + Altera o array original removendo o elemento.
* shift()
  + Retira o primeiro elemento do array.
  + Altera o array original removendo o elemento e trocando o índice de todos os elementos para um a menos do que eram, o índice 1 passa a ser o 0, o 2 passa a ser o 1, e assim por diante.
* push()
  + Adiciona o elemento passado como parâmetro do array, porém na última posição.
  + Altera o array original com o novo valor.
* unshift()
  + Funciona igual ao push(), porém adiciona na primeira posição e acaba trocando o índice de todos os elementos.
  + Altera o array original com o novo valor.
* reduce()
  + Utiliza uma função definida pelo usuário em cada um dos elementos, guardando o resultado em uma variável que pode ser acessada dentro da função que foi definida, retornando um único valor no final, reduzindo o array para um único valor.
* reduceRight()
  + Funciona igual o reduce() porém começa do final do array e segue até o início.
* reverse()
  + Inverte a ordem dos elementos do array, então o primeiro vira o último, o segundo o penúltimo e assim por diante.
* slice()
  + Copia uma parte do array para outro array.
* sort()
  + Organiza o array de acordo com a classificação Unicode, onde os números vêm antes das letras, porém não funciona corretamente para números, onde temos que definir uma função que irá auxiliar o comando.
* splice()
  + Consegue remover, um ou mais elementos consecutivos caso o segundo parâmetro tenha um valor maior que 0, e incluir um ou mais elementos a partir de um índice escolhido.

**Documentação array:** [**https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global\_Objects/Array**](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array)

**Dividindo com slice()**

Vai dividir uma lista com os determinados parâmetros que forem colocados

const sala1 = nomes.slice(0,nomes.lenght/2) – O parâmetro é o seguinte: (número que começa a startar, número que para a primeira lista)

const sala2 = nomes.slice(nomes.length/2) – aqui é a outra metade da lista

**Alterando com splice()**

splice() – remove algum elemento de qualquer local do array e permite colocar dentro desse array também

listaDeChamada.splice(1,2,’Rodrigo’) - O parâmetro é o seguinte: (qual índice vai começar, quantos elementos quer remover, qual elemento quer colocar nesse lugar). Para esse exemplo, vai tirar o elemento de índice 1, vai remover 2 elementos, e adicionar apenas a string.

**Concatenando arrays**

const salasUnificadas = sala1.concat(sala2); - dessa forma, irá concatenar a sala 1 com a sala 2, lembrando que temos que colocar uma nova constante para salvar a array com a concatenação

**Lista com 2 dimensões**

Let listaDeNotasEAlunos = [alunos, mediaDosAlunos]; – uma lista que contém listas dentro dela

console.log(`${listaDeNotasEAlunos[0][0]}, sua média é ${listaDeNotasEAlunos[1][0]{`) – nesse caso para o console log, devemos personalizar para o aluno e a média que queremos colocar.

Outra forma de chamar os arryas de 2 dimensões, é matrizes. Em que no excel podemos ver cada campo representando um elemento, e precisamos de uma linha e uma coluna para acessar um campo, então para as linhas são o primeiro colchete e a coluna o segundo colchete. E a Array, pode ter mais de 2 dimensões.

**Procurando na lista**

nomedalista.includes – vai ver se o que estamos procurando se está incluso na lista

nomedalista.indexOf – vamos buscar o índice do que?

**For clássico**

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

instrução do laço

}

**forEach**

um jeito diferente de passar por cada elemento da array, exemplo:

const notas = [10, 6.5, 8, 7.5];

let somaDasNotas = 0;

notas.forEach( nota => {

somaDasNotas += nota

)}

Let media = somaDasNotas/notas.length

console.log(media)

foreach – para cada, é um laço de repetição atual, ou seja, nesse exemplo, para cada nota, vamos fazer a somas das notas mais a nota. É um método callback, porque o parâmetro da função não é um dado, e sim outra função. Logo nesse exemplo ele faz a atribuição do i a cada vez que fazer o laço de repetição. Aqui não tem um retorno da função.

**Revisando Callbacks**

Funções callbacks – podemos fazer uma função de forma separada para fazermos por exemplo com mais de um array, ou simples fazer de forma separada, da seguinte forma:

const nomes = [“Ana”, “Ju”, “Leo”, “Paula”]

nomes.forEach(ImprimeNomes)

function ImprimeNomes(nome){

console.log(nome)

}

Dessa forma, criamos a nossa função que tem como parâmetro nome, e usamos o laço dentro do array contendo os nomes, então o mesmo vai imprimir nome por nome. E podemos colocar o índice também como parâmetro, da seguinte forma: nomes.forEach(ImprimeNomes, indice)

**Método .map**

Usamos o .map no array para adicionar um valor aos valores, da seguinte forma:

const notas = [10, 9, 8, 7, 6]

const notasAtualizadas = notas.map( nota => nota == 10 ? nota : ++nota)

console.log(notasAtualizadas)

Dessa forma, o .map vai alterar as notas com +1, e ainda utilizamos um if dentro do mesmo para que não ocorra que notas maiores que 10 sejam somadas.

**Alterando strings com map()**

toUpperCase – pega todas as letras da string e deixa maiúsculo

let nomes = [“ana Julia”, “Caio vinicius”, “BIA silva”]

const nomesAtualizados = nomes.map(nome => nome.toUpperCase())

console.log(nomesAtualizados)

- logo passamos a nossa array dos nomes, e no map colocamos a função para colocar cada nome com todas as letras maiúsculas. Logo o map() passa dentro da array e fazemos na função o que deve ser feito com os elementos dessa array. Para alteramos por exemplo só uma letra, podemos fazer o acesso do índice da letra pertencente a essa string, logo, vemos da seguinte forma:

const nome = “Alura”;

let nomeMaisculas = “”;

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

nomeMaisculas += nome[i].toUpperCase()

}

console.log(nomeMaisculas) // ALURA – nesse caso o map() pode retornar um array se a função informada tiver um return em seu escopo.

Para o JS, qualquer função que seja chamada como argumento de outra é considerada uma função call-back, não apenas em caso de métodos. Outros exemplos tem aqui: <https://developer.mozilla.org/pt-BR/docs/Glossary/Callback_function>

**Filtrando elementos**

filter – que é filtrar

const nomes = [‘Ana’, ‘Marcos’, ‘Maria’, ‘Mauro’]

const notas = [7, 4.5, 8, 7.5]

const reprovados = nomes.filter( (aluno,índice) => notas[índice] < 5)

console.log(`reprovados: ${reprovados}`);

Para esse filtro, e usando esse exemplo, vemos que ele filtra a array com os nomes, e vai verificar os parâmetro aluno e o índice, em que ele vai olhar a nota do aluno em um índice, e o filter, ele usa valores booleanos (true or false), logo, se a nota de um aluno em um determinado índice for menor que 5, ele vai colocar no novo array dos reprovados;

const reprovados = nomes.filter( (\_,índice) => notas[índice] < 5) – o \_ serve para dizer que o JS sabe que ele existe, mas que não vai ser usado no escopo da função, pois o filter, pega um dado dos nomes.

**Somando com reduce**

Reduzir – ou seja traz todo o array para um único valor númerico, em que o reduce precisa de uma função e dois parâmetros que um é o acumulador (acum) e o outro é valor atual (atual) – valor atual do looping, e o zero depois é o início para iniciar os valores dos array – os nomes acum e atual pode colocar qualquer nome. Exemplo

const notas = [10, 6.5, 8, 7];

const media = notas.reduce((acum, atual) => atual + acum, 0) / notas.length;

console.log(media);

em que o acum+atual é uma soma

Caso tenha mais de uma linha, tem que usar as chaves e o return