Javascript

Fechamento sempre com ponto vírgula no final ;

Criar variáveis apenas com let e const, pois as mesmas respeitam escopo de bloco

Utilizar Let apenas para variáveis que vão trocar valor

1. **JavaScript para Iniciantes**

**1.1 Tipos de dados VAR ou LET ou CONST**

**1.2 Tipo Strings**

1.2.a Aspas

1.2.b Template Strings

**1.3 Números**

1.3.a Operadores aritméticos

1.3.c Ordem aritmética importa

1.3.d incrementando 1

1.3.e String para Numero

**1.4 Boolean e Condicionais**

1.4.a Boolean

1.4.b Condicionais if e else,

1.4.c Condicional else if

1.4.d Valores que retornam verdadeiro ou falso; Truthy ou falsy

1.4.e Operador logico de negação

1.4.f Operadores de comparação

**1.5 Switch**

**1.6 Funçoes**

1.6.a Parâmetros e Argumentos

1.6.b Argumentos podem ser funções/ funções anônimas

1.6.c Verificando se o dado informado estã correto antes de executar uma função;

1.6.d Escopo

**1.7 Objetos**

1.7.a Metodos – functions / função em objetos

1.7.b Acessando a propriedade e atribuindo valor

1.7.c Incluindo propriedade a um objeto já criado

1.7.d this

1.7.e Tudo é objeto

1.7.f QuerySelector

**1.8 Array**

**1.9 Loop - for**

1.9.a Executando for com array

**2.0 While loop**

**2.1 Break**

**2.2 forEach**

**2.3 Operadores de atribuição**

2.3.a Operador ternário / Abreviação if e else

**2.4 Escopo**

2.4.a Const e Let

**3.0 Dom para iniciantes**

**3.1 DOCUMENT OBJECT MODEL (DOM)**

3.1.a addEventListner – Função anonima

3.1.b addEventListner – função separada

**3.2 Seleção de elementos**

3.2.a GetElementsById - único

3.2.b GetElementsByClassname - Retorna HTML Collection

3.2.c GetElementsByTagName - Retorna HTML Collection

3.2.d QuerySelector - único

3.2.e QuerySelectorAll - Retorna NODE list

3.2.f HTMLCOLLECTION vs NODELIST

3.2.g Array-like – foreach – Array from

3.2.h Arrow Function

**3.3 Adicionar remover ou alterar classes**

**3.4 Attributes**

3.4.a getAttribute setAttribute

3.4.b READ ONLY VS WRITABLE

**3.5 Dimenções e distancia**

3.5.a Height e width

3.5.b offsetTop e offsetLeft

3.5.c getBoundingClientRect()

3.5.d Window

3.5.e MatchMedia() – Verifica por JS a largura do navegador

**3.6 Eventos**

3.6.a addEventListener

3.6.b addEventListener > function > event

3.6.c event.preventDefault()

3.6.d This em addElementListener

3.6.e diferentes eventos de event listener;

3.6.f Foreach em eventlistener

**3.7 Traversing e Manipulação**

3.7.a OuterHTML – InnerHTML e InnerTEXT

3.7.b TRASVERSING

3.7.c Element vs Node

3.7.d Manipulando / movendo elementos

3.7.e Novos elementos – createElement()

3.7.f Clonando elementos

**3.8 Navegação por tabs**

3.8.a Só ocorre caso tenha javascript

**3.9 Acordeon list**

1. **VAR ou LET ou CONST**

* Para armazenar um valor a uma variável utiliza-se o “ = “ podendo no caso de números realizar equações;
* É possível também declarar varias variáveis utilizando apenas um = , separando por vírgula;
* Se utiliza Camel case nas variáveis, por exemplo totalComparado
* As variáveis podem iniciar com \_ ou $
* Sempre definir a variável no topo
* Let é uma variável que só pode ser declarado uma vez
* Const(constante) é uma variável que não pode ser redeclarada

*let* nome = 'jeziel';

*let* idade = '28';

console.log (nome, idade,'anos')

*let* \_preco = "20",

  $totalComparado = "5",

  $precoTotal = \_preco \* $totalComparado;

console.log("preço total é de: R$", $precoTotal);

jeziel 28 anos

preço total é de: R$ 100

* 1. **Tipos de dados**

Para verificar qual é o tipo de uma variável, utilizar o typeof;

console.log (typeof nome);

*var* nome = 'André'; *// String // palavras*

*var* idade = 28; *// Number // números, sem aspas*

*var* possuiFaculdade = true; *// Boolean // true or false*

*var* time; *// Undefined*

*var* comida = null; *// Null*

*var* simbolo = Symbol() *// Symbol*

*var* novoObjeto = {} *// Object*

**1.2 Tipo Strings**

É possível juntar duas strings, sendo ela de qualquer tipo, das seguintes formas;

*var* nome = 'Jeziel'; *// String*

*var* sobrenome = 'Silva'

*var* nomeCompleto = nome + ' ' + sobrenome;

*var* idade = 28; *// Number*

*var* frase = nomeCompleto + ' Idade ' + idade

console.log(frase)

**1.2.a Aspas**

O uso de aspas deve ser feito por um tipo apenas, ou ‘ ’ ou “ ”, Caso esteja usando “ ” e deseja mencionar algo com estas aspas, deve se utilizar backslash antes das aspas, ex;

*var* frase2 = "Esse é o \"melhor\" jogo"

**1.2.b Template String**

Para evitar o uso do +, podemos utilizar `${}` para passar expressões/variáveis ex;

*var* gols = 1000;

*var* frase1 = **`**Romario fez ${gols} gols**`**;

*var* frase1 = **`**Romario fez ${gols \* 2} gols**`**; *// Templete string possibilita essa multiplicação*

console.log(frase1)

*var* Pnome = 'Jeziel';

*var* Snome = 'Silva';

*var* nome12 = **`**${Pnome} ${Snome}**`**

obs.: aspas invertidas

**1.3 Numeros**

*var* idade = 28;

*var* gols = 1000;

*var* pi = 3.14; *// ponto para decimal*

*var* exp = 2e10; *// 20000000000 //exponencial, a qnt após o 'e' é a quantidade de 0 após o numero/ caso o numero após o e for negativo, ele adicina o zero a esquerda*

**1.3.a Operadores aritméticos**

*var* soma = 100 + 50; *// 150*

*var* subtracao = 100 - 50; *// 50*

*var* multiplicacao = 100 \* 2; *// 200*

*var* divisao = 100 / 2; *// 50*

*var* expoente = 2 \*\* 4; *// 16 // 2\*2\*2\*2=16*

*var* modulo = 14 % 5; *// 4 // resto da divisão, 10/5 - resto 4*

**1.3.b NaN**Caso tente somar string com number, o item se torna uma String, apenas se torna numero quando é uma subtração ou multiplicação

*var* numero = 80;

*var* unidade = 'kg';

*var* peso = numero + unidade; *// '80kg'*

*var* pesoPorDois = peso / 2 *// NaN (Not a Number)*

**1.3.c Ordem aritmética importa**

*var* total1 = 20 + 5 \* 2; *// 30 // Irá seguir automaticamente a ordem aritmetica, primeiro multiplicação ou divisão, depois subtração ou adição*

*var* total2 = (20 + 5) \* 2; *// 50 // é possível priorizar uma expressão, para sair da ordem aritmética*

*var* total3 = 20 / 2 \* 5; *// 50*

*var* total4 = 10 + 10 \* 2 + 20 / 2; *// 40*

**1.3.d Incrementando 1**

*var* x = 5;

console.log (x++) *// = 5 //Declara primeiro, depois incrementa*

console.log(x) *// = 6*

*var* y = 5;

console.log (++x) *// = 6 // incrementa e declara*

console.log(x) *// = 6*

*//Caso fosse utilizado o const, não seria possível fazer a operação, pois ele é inalterável*

*// o mesmo funciona para -- ou ++*

**1.3.e String para Numero**

*var* idade = +'28'; *// o sinal + transforma a string em number*

*// caso a string tenha apenas números, consegue realizar operações porem continua como string, mas caso tenha alguma letra se torna NaN*

**1.4 Boolean e Condicionais**

**1.4.a Boolean**

é sempre true ou false, exemplo de utilização utilizando IF e ELSE

**1.4.b Condicionais if e else,**

Se if for verdadeiro, tudo dentro do conchete será executado, caso contrario apenas o else;

*var* possuiGraduacao = true; *//caso seja true, ira retornar o if, caso false, retorna o else*

if(possuiGraduacao) {

  console.log('Possui graduação');

} else {

  console.log('Não possui graduação');

}

Caso for ocorrer apenas uma linha, podemos abreviar da seguinte forma;

*var* possuiFaculdade = true;

if(possuiFaculdade)

console.log('Possui Faculdade');

else

console.log('Não possui Faculdade');

**1.4.c Condicional else if**

*var* possuiGraduacao = true;

*var* possuiDoutorado = true;

*// caso o segundo seja verdadeiro, retorna o if*

if(possuiDoutorado) {

  console.log('Possui graduação e doutorado');

*// caso o segundo seja falso, retorna o else if*

} else if(possuiGraduacao) {

  console.log('Possui graduação, mas não possui doutorado');

*// caso o primeiro seja falso retorna o else*

} else {

  console.log('Não possui graduação');

}

**1.4.d Valores que retornam verdadeiro ou falso; Truthy ou falsy**

*// Falsy*

if(false)

if(0) *// ou -0*

if(NaN)

if(null)

if(undefined)

if('') *// ou "" ou ``*

*// Truthy*

if(true)

if(1)

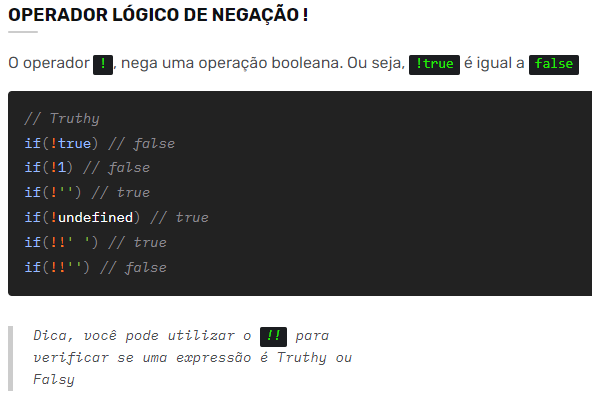
if(' ')

if('andre')

if(-5)

if({})

**1.4.e Operador logico de negação**



**1.4.f Operadores de comparação**

Retorna o valor booleano

**Operadores de maor ou menor**

10 > 5; *// true // maior que*

5 > 10; *// false // maior que*

20 < 10; *// false //  menor que*

10 <= 10 *// true menor ou igual*

10 >= 11 *// false maior ou igual*

Operadores de igual ou falso

**O == faz uma comparação não tão estrita e o === faz uma comparação estrita,** ou seja, o tipo de dado deve ser o mesmo quando usamos ===

*// para igual*

10 == '10'; *// true // levemente verdadeiro // string e number, porem iguais*

10 == 10; *// true // Levemente verdadeiro // dois number*

10 === '10'; *// false // estritamente falso // number e string*

10 === 10 *// true // estritamente verdadeiro // number e number*

*// para diferente*

10 != 15 *// true // levemente verdadeiro // dois numeros diferentes*

10 != '10' *// false // levemente falso //  dois numeros iguais, porem number e string*

10 !== '10' *// true // estritamente falso // iguais, porem number e string*

Operadores lógicos &&

true && false; *// false*

false && true; *// false*

'Gato' && 'Cão'; *// 'Cão'*

(5 - 5) && (5 + 5); *// 0*

'Gato' && false; *// false*

(5 >= 5) && (3 < 6); *// true*

*// Se ambos os valores forem true ele irá retornar o último valor verificado. Se algum valor for false ele irá retornar o mesmo e não irá continuar a verificar os próximos*

Operadores lógicos ||

true || true; *// true*

true || false; *// true*

false || true; *// true*

'Gato' || 'Cão'; *// 'Gato'*

(5 - 5) || (5 + 5); *// 10*

'Gato' || false; *// Gato*

(5 >= 5) || (3 < 6); *// true*

*//Retorna o primeiro valor true que encontrar*

**1.5 Switch**

Switch - opção caso e ou, Caso uma opção seja escolhida ela é selecionado, ou é verificado a próxima

*var* corFavorita = ''; *// valor recebido*

switch (corFavorita) { // Variavel a ser checada

  case 'Azul':*// caso azul essa é selecionado*

    console.log('Olhe para o céu.');

    break; //Caso verdadeira, para de verificar

  case 'Vermelho':

    console.log('Olhe para rosas.');

    break;

  case 'Amarelo':

    console.log('Olhe para o sol.');

    break;

  default:// caso nenhuma seja selecionado, irá mostrar essa

    console.log('Feche os olhos');

}

1.6 Funçoes

Blocos de código que podem ser executados e reutilizados, podem ser passados com ou sem valor, e pode retornar valores

Ex;

*function* areaQuadrado(*lado*) {*//função estabelecida // conteudo entre parenteses é uma var interna*

  return lado \* lado *//conteudo da função*

}

console.log(areaQuadrado(6))*//Chama a função e da valor // "6" no caso é o valor enviado área "lado"*

*function* res() {

  return 10 \* 3 *// =30*

}

*var* total = 5 \* res(); *// é possivel chamar uma função diretamente em equações, lembrando de chamar a função com parenteses " res() "*

console.log(total)

1.6.a Parâmetros e Argumentos

São os valores enviados para a execução da função

*function* imc(*peso*, *altura*) {*//peso e altura são os parametros, variaveis utilizadas dentro da função*

*let* imc = peso / (altura \*\* 2); *//calculo do imc*

  return imc;*//manda o imc de volta*

}

console.log(**`**Seu IMC é de ${imc(80, 1.80)}**`**) *//console chamando a função com os argumentos*

*function* corFavorita(*cor*) {

  if (cor == 'amarelo') { *// aspas pois é um texto*

    return "Você gosta do sol!"; *// retorna esse texto para a caixa onde veio a cor*

  } else if (cor == 'azul') {

    return "Você gosta do ceu!";

  } else if (cor == 'verde') {

    return "Você gosta da flotesta!";

  } else {

    return "Você não gosta de nenhuma cor"; *// caso nenhuma cor, enviara esse texto*

  }

}

console.log(**`**Que legal, ${corFavorita('')}**`**) *//aspas pois é um texto*

1.6.b Argumentos podem ser funções/ funções anônimas

addEventListener('click', *function*(){console.log('Clicou') });

*//addEventListener com o 'Click" executa uma função*

*// a função está comprimida na linha, mas é a mesma sintax*

1.6.c Verificando se o dado informado estã correto antes de executar uma função;

*function* terceiraIdade(*idade*) {

*//Caso o dado informado não for um numero, é enviado o return, e todo resto da função é ignorado*

  if (typeof idade !== 'number') {

    return 'por favor preencha um numero'

  } else if (idade >= 60) {

    return true;

  } else {

    return false;

  }

}

console.log(terceiraIdade('oi'))

**1.6.d Escopo**

Variaveis e funções definidas dentro de um bloco, não são visíveis fora dele

*function* precisoVisitar(*paisesVisitados*) {

*var* totalPaises = 193;

  return **`**Ainda faltam ${totalPaises - paisesVisitados} paises para visitar**`**

}

console.log(totalPaises); *// erro, totalPaises não definido*

*var* profissao = 'Designer';

*function* dados() {

*var* nome = 'André';

*var* idade = 28; *//Não utilizado porque na função da return repete*

*function* outrosDados() {

*var* endereco = 'Rio de Janeiro';

*var* idade = 29;

    return **`**${nome}, ${idade}, ${endereco}, ${profissao}**`**;

  }

  return outrosDados();

}

dados(); *// Retorna 'André, 29, Rio de Janeiro, Designer'*

outrosDados(); *// retorna um erro pois está dentro de outra função*

1.7 Objetos

Objeto é uma variável que pode ter diversas propriedades, por ex;

*var* pessoa = {

  nome: 'Jeziel',

  idade: 28,

}

*//para acessar:*

console.log (pessoa.nome)

**1.7.a Metodos – functions / função em objetos**

É um propriedade que possui uma função e não um valor

*var* quadrado = {

  lados: 4,

  area: *function*(*lado*) {

    return lado\*lado

  }

}

console.log(quadrado.area(4)) *// é chamado a var seguido da propriedade, mas podendo ser adicionado  um dado para a função*

**1.7.b Acessando a propriedade e atribuindo valor**

É localizada pelo nome da variável ponto o nome da propriedade, ex;

*var* menu = {

  width: 800,

  height: 200,

  backgroundcolor: 'black',

}

menu.backgroundcolor = 'white'

*var* bg = menu.backgroundcolor *// atribuindo a propriedade a uma nova var*

**1.7.c Incluindo propriedade a um objeto já criado**

Basta adicionar o nome do objeto ponto e a nova propriedade + o valor;

O mesmo serve para adicionar funções

menu.color = 'blue'

{width: 800, height: 200, backgroundcolor: 'white', color: 'blue'}

**1.7.d this**

O this é utilizado para mencionar o proprio objeto

*var* quadrado = {

  lados: 4,

  area: *function*(*lado*) {

    return lado\*lado

  },

  perimetro(*lado*){ // não é necessário a declaração function

    return this.lados \* lado *// a pripriedade this seleciona o objeto, e a propriedade seguido do ponto; this.lados*

  }

}

*var* height = 120;

*var* menu = {

width: 800,

height: 50,

metadeHeight() {

*return* *this*.height / 2;

} // com o this ele busca a variável/propriedade de dentro do objeto

}

menu.metadeHeight(); *// 25*

*// sem o this, seria 60*

**1.7.e Tudo é objeto**

Até mesmo uma variável simples é um objeto,

Que por exemplo tem a propriedade name.lenght;, que pode retornar o comprimento da variável;

*var* nome = 'André';

nome.length; *// 5*

nome.charAt(1); *// 'n'*

nome.replace('ré', 'rei'); *// 'Andrei' // substituir ré por rei*

nome; *// 'André'*

**1.7.f QuerySelector**

*var* btn = document.querySelector('#botao');

*// com essa função podemos criar uma variavel, ebuscar no documento com o QuerySelector*

*// a class com " . " ou o Id com " # " a classe ou id desejado, e atribuir a var*

btn.addEventListener('click', *function* clicou(){console.log('clicou')})

*//Dessa forma, podemos por ex adicionar um event a var*

btn.classList.add('ativo')

*// ou adicionar uma classe*

**1.8 Array**

É um grupo de valores geralmente relacionados. Servem para guardarmos diferentes valores em uma única variável, acessíveis por números, ex;

*var* videoGames = ['Switch', 'PS4', 'XBox'];*// Array criada entre conchetes*

*//videoGames[0] // Switch*

*//videoGames[1] // PS4*

*//videoGames[2] // Xbox*

*//exemplo de metodos e propriedades*

*/\*videoGames.pop(); // Remove o último item e retorna ele*

*videoGames.push('3DS'); // Adiciona ao final da array*

*videoGames.length; // 3\*/*

*var* ultimogame = videoGames.pop() *// inclui o ultimo valor na variavel, e exclui o mesmo*

*/\**

*videoGames*

*(2) ['Switch', 'PS4']*

*ultimogame*

*'XBox'*

*\*/*

**1.9 Loop - for**

Repetições for - para

*// for (para) var que é igual a zero, for menor que 10, executa e recebe mais 1*

*// verificação no inicio*

for (*var* num = 0; num <= 10; num++) {

*//ação a ser executada*

  console.log(num);

}

**1.9.a Executando for com array**

*var* videoGames = ['Switch', 'PS4', 'XBox', '3DS']

*//enquanto item for menor que o comprimento da array, item recebe +1*

for (*var* item = 0; item < videoGames.length; item++ ) {

  console.log(videoGames[item]) *// conforme item recebe mais um, muda o item da array*

**2.0 While loop**

Repetoções while - enquanto

*//while(enquanto) i for maior ou igual a zero, i recebe -1*

*var* i = 10

while (i >= 0) {

  console.log(i);

  i--;

}

*//while a for menor ou igual a 10 a recebe ele mesmo mais 2*

*var* a = 0

while (a <= 10) {

  console.log(a);

  a = a + 2;

}

**2.1 Break**

Para parar um loop caso ele encontre um valor especifico;

*var* videoGames = ['Switch', 'PS4', 'XBox', '3DS']

*//enquanto item for menor que o comprimento da array, item recebe +1*

for (*var* item = 0; item < videoGames.length; item++ ) {

  console.log(videoGames[item]) *// conforme item recebe mais um, muda o item da array*

  if (videoGames[item] === 'XBox') {

    break;

loop.js:22 Switch

loop.js:22 PS4

loop.js:22 Xbox

**2.2 forEach**

forEach é um método de Array, alguns objetos array-like possuem este método. Caso não possua, o ideal é transformá-los em uma array.

Ex; 01

*const* titulos = document.getElementsByClassName('titulo');

*const* titulosArray = Array.from(titulos);

titulosArray.forEach(*function*(*item*){

  console.log(item);

});

Ex 2

*//foreach chama uma função, para cada item da array*

videoGames.forEach(*function*(*item*){

  console.log(item)

})

*//*

frutas.forEach(*function*(*item*, *index*){

  console.log(item, index)

})

loop.js:34 maça 0

loop.js:34 banana 1

loop.js:34 pera 2

loop.js:34 abacaxi 3

**2.3 Operadores de atribuição**

São operadores de atribuição assim como “ = “

Significa por exemplo que o item recebe ele mesmo + outro, ex;

*var* x = 5;

*var* y = 10;

x += y; *// x = x + y (15)*

x -= y; *// x = x - y (-5)*

x \*= y; *// x = x \* y (50)*

x /= y; *// x = x / y (0.5)*

x %= y; *// x = x % y (0)*

x \*\*= y; *// x = x \*\* y (9765625)*

**2.3.a Operador ternário / Abreviação if e else**

Abrevia o if e else em apenas uma linha, na atribuição, caso a condição for true ou false, atribui os seguintes valores na variável;

*var* idade =  17

*var* podebeber = (idade >= 18) ? 'Pode beber' : 'Não pode beber';

console.log(podebeber)

*var* idade =  18

naoPossuiDiabetes = true

*var* podebeber = (idade >= 18 && naoPossuiDiabetes) ? 'Pode beber' : 'Não pode beber';

console.log(podebeber)

*// Podemos verificar duas ou mais confirmações com true ou false*

*=\*=*

*// para uma verificação para atribuição, dessa forma irá atribuir um valor booleano*

*var* possuiCarro = true;

*var* possuiCasa = true;

*var* darCredito;

darCredito = (possuiCarro && possuiCasa);

console.log(darCredito)

**2.4 Escopo**

*/\**

*Variáveis criadas com var, vazam o bloco. Por isso com a introdução*

*o ES6 a melhor forma de declarmos uma variável é utilizando const e let,*

*pois estas respeitam o escopo de bloco.*

*\*/*

*//Variáveis criadas com var, vazam o bloco. Por isso com a introdução do*

*//ES6 a melhor forma de declarmos uma variável é utilizando const e let,*

*//pois estas respeitam o escopo de bloco.*

{

*var* carro = 'Fusca';

*const* ano = 2018;

} *// exemplo de var vazando o bloco*

console.log(carro); *// Carro*

console.log(ano); *// erro ano is not defined*

**2.4.a Const e Let**

Const são variáveis que não permitem a modificação no valor

*const* data = {

  dia: 20,

  ano: 2023,

}

data.dia = 29 *// Possivel alterar propriedades do objeto const*

console.log(data)

*const* nome = 'jeziel'

console.log(nome)

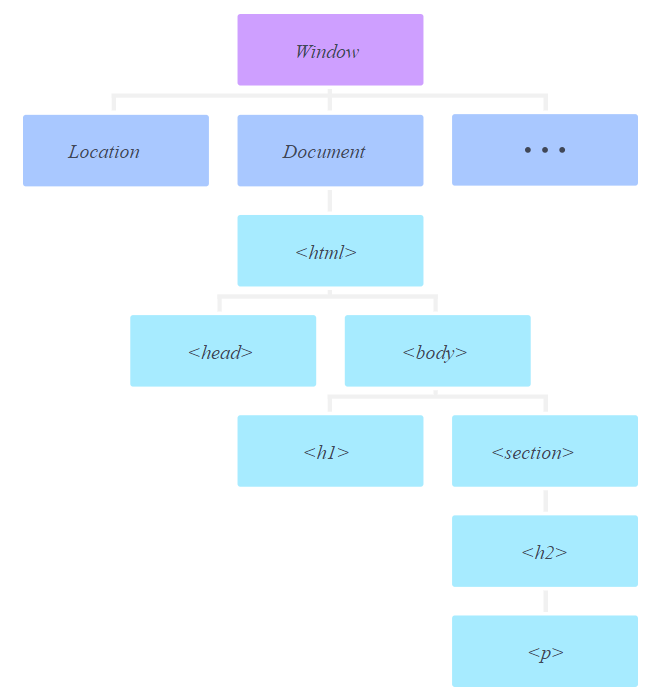
*const* nome = 'carlos' *// impossivel alterar o valor de uma const*

Let permite alteração

**3.0 Dom para Iniciantes**

**3.1 DOCUMENT OBJECT MODEL (DOM)**

Dom é a representação de toda a janela, por objetos, tudo tem propriedades e metodos



Todo objeto possui suas propriedades padrão, e podemos acessar sub propriedades com elas, como;

*const* titulo = document.querySelector('h1'); *// seleciona o h1 dentro do document*

titulo.innerText; *// retorna o texto;*

titulo.classList; *// retorna as classes; // para selecionar uma classe especifica poderíamos utilizar classList[2] entre aspas a ordem da classe*

titulo.id; *// retorna o id;*

titulo.offsetHeight; *// retorna a altura do elemento;*

**3.1.a addEventListner – Função anonima**

Utilizando o dom para selecionar um item, podemos adicionar um evento com o function

*const* tituloh1 = document.querySelector('#tit') *//seleciona a id #tit*

tituloh1.addEventListener('click', *function*(){

alert('clicou');

})

**3.1.b addEventListner – função separada**

Para separar a função, colocamos o nome da função após a ação;

*const* tituloh1 = document.querySelector('#tit') *//seleciona a id #tit*

tituloh1.addEventListener('click', callback1)

*function* callback1(){

  alert('clicou');

  }

**3.2 Seleção de elementos**

**3.2.a GetElementsById - único**

Seleciona apenas o elemento do ID

*const* animais = document.getElementById('animais')

**3.2.b GetElementsByClassname - Retorna HTML Collection**

Seleciona todos os elementos que tenham essa classe, podendo selecionar um especifico colocando entre conchetes

Nome em name da classe não começa com ponto

*const* gridSection = document.getElementsByClassName('grid-section')

console.log(gridSection[2])

**3.2.c GetElementsByTagName - Retorna HTML Collection**

Seleciona todos os elementos com a tag relacionada

*const* ul = document.getElementsByTagName('ul');

**3.2.d QuerySelector - único**

Seleciona o primeiro elemento que combinar com o seletor CSS

*const* animais = document.querySelector('.animais');

*const* contato = document.querySelector('#contato');

*const* ultimoItem = document.querySelector('.animais-lista li:last-child');

*const* linkCSS = document.querySelector('[href^="https://"]');

*const* primeiroUl = document.querySelector('ul');

*// Busca dentro do Ul apenas*

*const* navItem = primeiroUl.querySelector('li');

**3.2.e QuerySelectorAll - Retorna NODE list**

Seleciona todos os itens que tiverem a classe ou tag especificada

*const* animaistag = document.querySelectorAll('.animais img');

console.log(animaistag) *// retorna uma nodelist*

console.log(animaistag[2]) *// retorna o item especifico*

*// para retornar itens que começam com as iniciais*

*const* imagem1 = document.querySelectorAll('[src^="img/imagem"]')

console.log(imagem1)

*const* linkCSS = document.querySelectorAll('[href^="#"]');

console.log(linkCSS)

*// Selecione um paragrafo pelo lenght*

*const* ultimop = document.querySelectorAll('p');

console.log(ultimop.length)

console.log(ultimop[ultimop.length -1]);

**3.2.f HTMLCOLLECTION vs NODELIST**

A diferença está nos métodos e propriedades de ambas. Além disso a NodeList retornada com querySelectorAll é estática, html collection em tempo real

*const* titulo = document.querySelector('.titulo');

*const* gridSectionHTML = document.getElementsByClassName('grid-section');

*const* gridSectionNode = document.querySelectorAll('.grid-section');

titulo.classList.add('grid-section');

console.log(gridSectionHTML); *// 4 itens*

console.log(gridSectionNode); *// 3 itens*

**3.2.g Array-like – foreach – Array from**

HTMLCollection e NodeList são array-like, parecem uma array mas não são. O método de Array forEach() por exemplo, existe apenas em NodeList.

*const* imgs = document.querySelectorAll('img')

*let* i = 0;

imgs.forEach(*function*(*item*, *index*, *array*){ *// 1° para cada item 2° mostra o indice 3° mostra toda a lista*

console.log(item, index, array)

});

*// para transformar HTML Collection em array, comando Array.from;*

*const* titulos = document.getElementsByClassName('titulo'); *// pega o elemento em uma variavel*

*const* titulosArray = Array.from(titulos)  *// transforma em array*

titulosArray.forEach(*function*(*item*, *index*){ *//liberado para utilizar por ex o foreach*

  console.log(item, index)

})

Para por exemplo item, temos propriedades coo innettext;

p.forEach((*item*, *index*) =>   console.log(item.innerText))

**3.2.h Arrow Function**

Sintaxe curta em relação a function expression. Basta remover a palavra chave function e adicionar a fat arrow => após os argumentos.

Ex 1

*const* imgs = document.querySelectorAll('img');

imgs.forEach((*item*) => {

  console.log(item);

});

Ex 2

*const* imgs = document.querySelectorAll('img')

*let* i = 0;

imgs.forEach((*item*, *index*, *array*) => {

  console.log(item, index, array)

})

*// caso não for passar parametro, ainda precisa dos parenteses*

*// let i = 0;*

*// imgs.forEach(() => {*

*// console.log(i++)*

*//Podemos reduzir ainda mais caso for uma expressão simples, removendo o parenteses*

imgs.forEach(() => console.log(i++))

p.forEach((*item*, *index*) => console.log(item.innerText))

**3.3 Adicionar remover ou alterar classes**

*const* menu = document.querySelector('.menu') *// seleciona o grupo de classes*

menu.classList.add('ativo') *// adiciona classe ativo*

console.log(menu.classList)

menu.classList.remove('azul')*// remove a classe azul*

console.log(menu.classList)

menu.classList.toggle('azul')*// alternar classes, adiciona e remove a classe de um elemento*

console.log(menu.classList)

Exemplo de adicionar e remover classes

*// Adicione a classe ativo a todos os itens do menu*

*const* menuItens = document.querySelectorAll('.menu li') *// seleciona todas as li do menu*

console.log(menuItens) *// verifica*

menuItens.forEach(*function*(*item*){ *// para cada ITEM do menu,*

  item.classList.add('ativo') *// adiciona a classe*

})

console.log(menuItens) *// verifica*

*// Remove a classe ativo de todos os itens do menu e mantenha apenas no primeiro*

menuItens.forEach(*function*(*item*){  *// para cada ITEM do menu*

  item.classList.remove('ativo')  *// Remove o ativo*

})

console.log(menuItens) *// verifica*

menuItens[0].classList.add('ativo') *// Para o ITEM ZERO é adicionado a classe ativo*

console.log(menuItens) *// verifica*

**3.4 Attributes**

Retorna uma arraylike com os atributos do elementro

*const* animais = document.querySelector('.animais')

console.log(animais.attributes) *// retorna a arraylike*

console.log(animais.attributes.id) *// retorna o id do elemento*

console.log(animais.attributes.class) *// retorna a classe do elemento*

console.log(animais.attributes['data-texto'])*// caso tenha hifen, podera ser passado como string, ou por numero*

**3.4.a getAttribute setAttribute**

Adiciona, Seta, Remove ou verifica se um elemento tem um atributo

*//get attribute - retorna o atributo selecionado*

*const* img = document.querySelector('img'); *// puxa determinado elemento*

console.log(img)

*const* altImg = img.getAttribute('alt') *// retorna determinado atributo do elemento*

console.log(altImg)

*// setattribute - seta atributo e valor desejado, puxar pruimeiro o elemento em uma var*

img.setAttribute('alt', 'É uma raposa') *// seta nome do atributo + valor*

console.log(altImg)

*//hasattribute - verifica se um elemento tem atributo*

*const* possuiAlt = img.hasAttribute('alt')

console.log(possuiAlt)

//removeattribute

img.removeAttribute('alt')

Exemplo de set attribute;

*const* linkMenu = document.querySelector('.menu [href^="https://"]'); *// seleciono o link*

console.log(linkMenu); *//verifico*

linkMenu.setAttribute('href', '#'); *// seto o atributo na variavel do link*

Exemplo de has atribute;

*// Verifique se as imagens possuem o atributo alt*

*const* possuiAlt = img.hasAttribute('alt')

console.log(possuiAlt)

**3.4.b READ ONLY VS WRITABLE**

Existem propriedades que não permitem a mudança de seus valores, essas são considerados Read Only, ou seja, apenas leitura.

**3.5 Dimenções e distancia**

**3.5.a Height e width**

Estas são propriedades e métodos dos objetos Element e HTMLElement, a maioria delas são Read Only

*const* section = document.querySelector('.animais-lista');

*const* altura = section.clientHeight

*const* alturaScroll = section.scrollHeight

section.clientHeight

370

section.scrollHeight

656

*//section.clientHeight; // height + padding*

*//section.offsetHeight; // height + padding + border*

*//section.scrollHeight; // height total, mesmo dentro de scroll*

*// mesmo serve para width*

**3.5.b offsetTop e offsetLeft**

*const* animaisTop = section.offsetTop; *// Distância entre o topo do elemento e o topo da página*

console.log(animaisTop)

*const* animaisLeft = section.offsetLeft; *// Distância entre o canto esquerdo do elemento e o canto esquerdo da página*

console.log(animaisLeft)

Exercicio verificando se o link possui altura especificada;

*let* links = document.querySelectorAll('a') *// seleciona todos os links*

console.log(links)

links.forEach(*function*(*itens*) { *// para cada item do link*

if (itens.offsetWidth > 48 && itens.offsetHeight > 28 ) { *// ira verificar os parametros*

 console.log('é tamanho ocerto') *// se true*

} else {

  console.log('não é tamanho certo') *//se false*

}

})

*// console*

*/\**

*NodeList(4) [a, a, a, a] // são 4 links*

*script.js:76 é tamanho ocerto // retorna certo*

*script.js:78 não é tamanho certo // retorna o segundo, que esta errado*

*2 script.js:76 é tamanho ocerto // retorna o restante certo*

*console.log(links[1]) // puzando o segundo, que deu erro*

*script.js:81 <a href=​"#faq">​Faq​</a>​ // o item errado*

*\*/*

*// Se o browser for menor que 720px,*

*// adicione a classe menu-mobile ao menu*

*Exercicio para somar as larguras*

*// Retorne a soma da largura de todas as imagens*

console.log('Exercicios')

*let* soma = 0 *// variavel de contagem*

*function* somaImagens() { *// função que sera executada depois do load*

*const* Imagens = document.querySelectorAll('img') *// selecionando todas img*

Imagens.forEach(*function*(*item*){ *// para cada imagem*

  soma += item.offsetWidth *// irá somar as larguras*

  return soma *// retornando o valor da soma*

})

console.log(soma)*// retornando o valor da soma*

}

window.onload = *function*() { *// quando carregar todas as imagens*

  somaImagens();

}

**3.5.c getBoundingClientRect()**

Método que retorna um objeto com valores de width, height, distâncias do elemento e mais.

*const* primeiroH2 = document.querySelector('h2');

console.log(primeiroH2)

*const* rect = primeiroH2.getBoundingClientRect();

console.log(rect) *// retorna todas as cordenadas do objeto*

console.log(rect.top) *// mesmo que offsetTop*

Uma possível verificação se um elemento por exemplo deixou de ser exibido com o scroll, podemos utilizar o top;

if (rect.top < 0) {

  console.log('Passou do elemento')

}

**3.5.d Window**

console.log(

window.innerWidth, *// width do janela*

window.outerWidth, *// soma dev tools também*

window.innerHeight, *// height do janela*

window.outerHeight, *// soma a barra de endereço*

window.pageYOffset, *// distância total do scroll vertical*

window.pageXOffset, *// distância total do scroll horizontal*

)

**3.5.e MatchMedia() – Verifica por JS a largura do navegador**

*const* small = window.matchMedia('(max-width: 600px)');

if(small.matches) {

  console.log('Tela menor que 600px')

} else {

  console.log('Tela maior que 600px')

}

// outro exemplo para incluir classe

*const* menu1 = document.querySelector('.menu')

*const* navegador = window.outerWidth

*const* small1 = window.matchMedia('(max-width:720px)');

if (small.matches) {

  menu1.classList.add('menu-mobile')

}

**3.6 Eventos**

**3.6.a addEventListener**

Adiciona uma função ao elemento, esta chamada de callback, que será ativada assim que certo evento ocorrer neste elemento.

Forma simples com função anonima;

*const* img = document.querySelector('img')

console.log(img)

img.addEventListener('click', *function*(){

  console.log('Clicou')

})

Separando a função do evento, chame sem os ()

*const* img = document.querySelector('img')

console.log(img)

img.addEventListener('click', callclick) *// IMPORTANTE, apenas chame a função sem os parenteses*

*function* callclick(){

  console.log('Clicou')

}

**3.6.b addEventListener > function > event**

Podemos também pegar informações detalhadas, utilizando a tag event (ou qualquer descrição) junto a função;

*function* callclick(*event*){

  console.log('Clicou')

  console.log(event)

}

Segue algumas das propriedades do event;

*const* animaisLista = document.querySelector('.animais-lista');

animaisLista.addEventListener('click', executarCallback);

*function* executarCallback(*event*) {

*const* currentTarget = event.currentTarget; *// passa o valor da lista*

*const* target = event.target; *// qual item o clique ocorreu*

*const* type = event.type; *// tipo de evento // click*

*const* path = event.path;

  console.log(currentTarget, target, type, path);

}

**3.6.c event.preventDefault()**

Previne o comportamento padrão do evento no browser. No caso de um link externo, por exemplo, irá previnir que o link seja ativado.

*const* linkExterno = document.querySelector('a[href^="http"]');

console.log(linkExterno)

linkExterno.addEventListener('click', handleLinkExterno);

*function* handleLinkExterno(*event*) {

  event.preventDefault()

  console.log(event)

}

**3.6.d This em addElementListener**

Nesse caso, a palavra this se refere ao elemento que chamou o evento;

  console.log(this)

Geralmente igual ao event.currentTarget

**3.6.e Diferentes eventos de event listener;**

<https://developer.mozilla.org/en-US/docs/Web/Events>

Como os de mouse,

h1.addEventListener('click', callback);

h1.addEventListener('mouseenter', callback);

de janela;

window.addEventListener('scroll', callback);

window.addEventListener('resize', callback);

window.addEventListener('keydown', callback);

e com o event conseguimos adicionar ações ao apertar teclas do teclado;

window.addEventListener('keydown', hancleKeyboard);*// propriedade de aperto de tecla chamando a função*

*function* hancleKeyboard(*event*) {

  if (event.key === 'f') { *// função acionada pela tecla do parametro*

    document.body.classList.toggle('fullscreen')*// ação realizada*

  }

}

**3.6.f Foreach em eventListener**

Podemos chamar uma função para cada item da lista com o for each jogando para uma function;

*const* imgs = document.querySelectorAll('img');

  imgs.forEach((*img*) => {

    img.addEventListener('click', imgSrc);

  });

*function* imgSrc(*event*) {

*const* src = event.currentTarget.getAttribute('src');

    console.log(src);

  }

Exemplo removendo e adicionando classes a um elemento;

*const* links = document.querySelectorAll('a[href^="#"]')

links.forEach((*li*) => {

  li.addEventListener('click', clicou)

  })

*function* clicou(*event*) {

    event.preventDefault()

    console.log(event)

    links.forEach((*linksoff*) => {

    linksoff.classList.remove('ativo')

    })

    this.classList.add('ativo')

}

Exemplo removendo um item (this) clicado

*const* tudo = document.querySelectorAll('body \*')

tudo.forEach((*e*) =>{

  e.addEventListener('click', clickT)})

*function* clickT(*ev*) {

  ev.currentTarget.remove();

}

**3.7 Traversing e Manipulação**

**3.7.a OuterHTML – InnerHTML e InnerTEXT**

console.log(h1.innerHTML)  *// todo o html do elemento*

console.log(h1.outerHTML) *// todo o html do elemento*

console.log(h1.innerText) *// entrega o HTML apenas com o texto*

**3.7.b TRASVERSING**

Como navegar pelo DOM, utilizando suas propriedades e métodos.

lista.parentElement; *// pai*

lista.parentElement.parentElement; *// pai do pai*

lista.previousElementSibling; *// elemento acima*

lista.nextElementSibling; *// elemento abaixo*

lista.children; *// HTMLCollection com todos os filhos do elemento em questão,*

lista.children[0]; *// HTMLCollection que pode ser selecionado itens especificos como array[]*

lista.children[--lista.children.length]; *//  ou -1 pegamos o ultimo elemento da lista*

lista.querySelectorAll('li'); *// todas as LI's com queryselectorall*

lista.querySelector('li:last-child'); *// selecionando o ultimo com seletor CSS*

**3.7.c Element vs Node**

Element's represetam um elemento html, ou seja, uma tag. Node representa um nó, e pode ser um elemento (Element), texto, comentário, quebra de linha e mais.

*const* lista = document.querySelector('.animais-lista');

lista.previousElementSibling; *// elemento acima*

lista.previousSibling; *// node acima, pode ser um comentario ou até um espaço*

lista.firstChild; *// primeiro node child pode ser um comentario ou até um espaço*

lista.childNodes; *// todos os node child pode ser um comentario ou até um espaço*

**3.7.d Manipulando / movendo elementos**

contato.appendChild(lista); *// move lista para o final de contato*

contato.insertBefore(lista, titulo); *// insere a lista antes de titulo*

contato.removeChild(titulo); *// remove titulo de contato*

contato.replaceChild(lista, titulo); *// substitui titulo por lista*

*const* lista = document.querySelector('.animais-lista');

*const* contato = document.querySelector('.contato');

*const* titulo = contato.querySelector('.titulo');

*// move lista para o final da area contato*

contato.appendChild(lista);

*// na area contato, adiciona a lista antes do titulo de contato, o elemento onde vai inserir o before tem que ser filho da area*

contato.insertBefore(lista, titulo)

*// Remove um filho*

contato.removeChild(titulo); *// remove titulo de contato*

*// Substitui um elemento por um filho do pai*

contato.replaceChild(lista, titulo); *// substitui titulo por lista*

**3.7.e Novos elementos – createElement()**

Podemos criar novos elementos com o método createElement()

*const* novoh1 = document.createElement('h1') *// cria a variavel com a tag h1*

novoh1.innerText = 'Novo titulo'; *// adiciona texto ao h1*

novoh1.classList.add('titulo') *// da classe de titulo ao h1*

console.log(novoh1)

contato.appendChild(novoh1) *// adiciona o novo titulo no fim da area contato*

**3.7.f Clonando elementos**

Todo elemento selecionado é único. Para criarmos um novo elemento baseado no anterior, é necessário utilizar o método cloneNode()

*const* titulo = document.querySelector('h1'); *// seleciono o elemento a ser clonado*

*const* cloneTitulo = titulo.cloneNode(true); *// metodo de clone com true, assegurando que tosos os filhos sejasm clonadas*

*const* contato = document.querySelector('.contato');  *//seleciono a area onde vai ser inserido*

contato.appendChild(cloneTitulo);*// inserindo o clone em uma area*

**3.8 Navegação por tabs**

Nessesita ter duas section, uma com a sua lista clicavel, e a outra que irá aparecer conforme clica,

Cada lista tem seu index, sua classificação [0] [1] ...

Irá utilizar dessa ordem para alinhar os itens, por exemplo [0] raposa em uma lista, a outra [0] tem que ser a descrição da raposa

*//Navegação por tabs*

*// Primeiro selecionamos a lista que será clicada*

*const* tabMenu = document.querySelectorAll('.js-tabmenu li');

*//Depois a lista que irá mostrar/ocultar o conteudo*

*const* tabContent = document.querySelectorAll('.js-tabcontent section')

*// aqui nos deixamos o primeiro item por padrão, para não iniciar vazia*

tabContent[0].classList.add('ativo')

*//coloque toda a verificação dentro de uma função verificando se as variaveis tem lenght,*

*//codigo só ocorre se as duas seções tiverem comprimento, forem verdadeiras*

if(tabMenu.length && tabContent.length) {

*//começando com o foreach para dar cada um da lista o evento click, pegando junto o index*

*/\*'item' é o parametro interno, 'index' é o numero exato do item que foi ativdo\*/*

  tabMenu.forEach((*item*, *index*) => {

    item.addEventListener('click', () => { *// ao evento do click*

      activeTab(index); *//ativa a função activeTab enviando o numero do index /\*Index é o numero exato do item que foi ativado\*/*

    })

  });

*//crie uma classe com display none, para ser adicionada aos elementos que não tiverem sendo utilizados*

*//função responsavel por adicionar e remover a classe*

*function* activeTab(*index*) { *// ativa a função com o parametro index recebido*

    tabContent.forEach((*section*) => { *// para cada sessão /\*section é um nome escolhido para o parametro da variavel tabContent\*/*

      section.classList.remove('ativo') *// ele remove a classe ativo de todos*

    })

    tabContent[index].classList.add('ativo') *// e adiciona a classe ativo para o item da array(index) recebido*

  }

}

**3.8.a Só ocorre caso tenha javascript**

caso queira que o conteúdo apareça caso não tem JS

cadicionar uma classe ao HTML utilizando Javascript no head;

  <script>document.documentElement.className += ' js'</script> // adicione a classe contatenando um espaço, para que adicione da forma correta

E na classe que da display none, adiciona anterior a ela a classe que esta adicionando por js em html(elemento pai)

 .js .js-tabcontent section {

    display: none;

  }

Dessa forma, só ativara display none quando a classe não for ativada, pois a classe js não ira existor no html

**3.9 Acordeon list**

Técnica utilizada para ocultar uma resposta até que seja clicado no link superior;

*// Accordionlist - Utilizado para ocultar, como por ex a descrição de uma FAQ*

*//desclarando todo o conteudo em uma função, e chamando ela no final*

*function* initAccordion() {

*//selecione toda a sessão onde está a pergunta + a pergunta, nesse caso o dt*

*const* accordionList = document.querySelectorAll(".js-accordion dt");

*// declaração de variavel com o nome da classe a ser empregada*

*const* activeClass = 'ativo'

*// dando classe ao primeiro item para iniciar aberto*

  accordionList[0].classList.add(activeClass);

  accordionList[0].nextElementSibling.classList.add(activeClass);

*//acom o if, a função só ocorrerá se gouver lenght*

  if (accordionList.length) {

*// aqui nos vamos passar um por um verificando o click, caso houver ativara a function*

    accordionList.forEach((*item*) => {

      item.addEventListener("click", activeAccordion);

    });

*//caso ative, a pergunta e o seguinte recebera a classe ativo*

*function* activeAccordion() {

      this.classList.toggle(activeClass);

      this.nextElementSibling.classList.toggle(activeClass);

    }

  }

}

initAccordion(); *//inicia a accordion*

4.0 Scroll Suave

Utilizado para quando o usuário clicar em um link interno, seja redirecionado com scroll suave

*//vamos iniciar tudo dentro de uma function,*

*function* initScrollSuave() {

*//scroll suave*

*// para iniciar, damos uma classe js para o menu, selecionando apenas os itens que são links internos,*

*const* linksInternos = document.querySelectorAll('.js-menu a[href^="#"]')

*//criamos um foreach para cada um passar pela function ao clique*

linksInternos.forEach((*link*) => {

  link.addEventListener('click', scrollToSection);

});

*// aqui cada um vai ter o padrão prevenido*

*function* scrollToSection(*event*) {

  event.preventDefault();

*// então vamos ccolocar em href o exato atributo que foi clicado*

*const* href = event.currentTarget.getAttribute('href');

  console.log(href)

*// para linkar o id do link com a sessão, vamos utilizar o query selector com a var href*

*const* section = document.querySelector(href);

  console.log(section);

*// agora já temos o link que foi clicado, e a sessão pertencente*

*// vamos utilizar o window.scrollTo para ir até onde está a section,*

*// Podemos utilizar o scroll into view, um metodo como objeto*

  section.scrollIntoView({

    behavior: 'smooth', *// suave*

    block: 'start', *// inicio da section*

  })

*// utilizando o intoview com o block start, não precisamos pegar o offset top*

*/\* MODO ALTERNATIVO*

*//mas primeiro vamos pegar o topo de cada sessão com o offset*

*const topo = section.offsetTop;*

*//Para utilizarmos o scroll to, criamos um objeto com o top com a var da sessão, e o comportamento como smooth*

*window.scrollTo({*

*top: topo,*

*behavior:"smooth"*

*})  \*/*

}

}

initScrollSuave() *// chamando a function*