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

3.2.i – Arrow Function Reduzido

**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**

**4.0 Scroll Suave**

**5.0 Objetos**

**5.1 Constructor Function – Função Construtora**

**5.2 Prototype**

5.2.a Entendendo o que está retornando;

5.2.b Verificando os metodos

**5.3 Objects - Native, host e user**

**5.4 Metodos e propriedades da STRING**

5.4.a Length

5.4.b Charat

5.4.c Contatenando Strings

5.4.d Includes

5.4.e Startswith Endswith

5.4.f Slice

5.4.g Indexof e Lastindexof

5.4.h Padstart

5.4.i Repeat

5.4.j Replace

5.4.k Split e Join

5.4.l toUpperCase e toLowerCase

5.4.m Trim

**5.5 Number**

5.5.a verificar se é numero isNaN

5.5.b isInteger verifica se é integral

5.5.c ParseFloat

5.5.d ParseInt

5.5.e toFixex

5.5.f toString

5.5g LocalString

**5.6 Math**

5.6.a Arredondamento de números

5.6.b Min e max

5.6.c Gera numero aleatório

5.6.d tres pontos para verificar um math

**6.7 Array**

6.7.a Construção de Arrays

6.7.b Array From – transforma em array

6.7.c IsArray – verifica se é array

6.7.d Criação de array - Array.of Array() e New Array()

6.7.e Length – tamanho da array

6.7.f Sort

6.7.g Unshift e Push – adicionando e removendo elementos

6.7.h Shift e Pop Remove itens

6.7.i Reverse – invertendo a ordem

6.7.j Splice adicionando em index

6.7.k Copy within

6.7.l Fill() – preencher

6.7.m Concat – Contatenar

6.7.n Inclues, Indexof e Lastindexof

6.7.o Join - Juntar valores Split – Separar

6.7.p Split separa

6.7.q Slice Corta pedaço

6.7.r clonar array

**6.8 Metodos de interação objetos Array**

6.8.a Diferenças entre Array e MAP

6.8.b ForEach

6.8.c Modificando a array original com o foreach

6.8.d Arrow Function

6.8.e Map

6.8.f Criando uma const com uma função embutida

**6.9 Reduce – Acumulador**

6.9.a Reduce Equações

6.9.b Reduce maior ou menor numero

6.9.c Reduce passo a passo

6.9.d Retornando propriedades de objetos com reduce

6.9.e Reduce Right

**7.0 Some**

**7.1 Every**

**7.2 Find e Find Index**

7.2.a Find

7.2.b FindIndex

**7.3 Filter**

7.3.a Filter em Objetos

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)

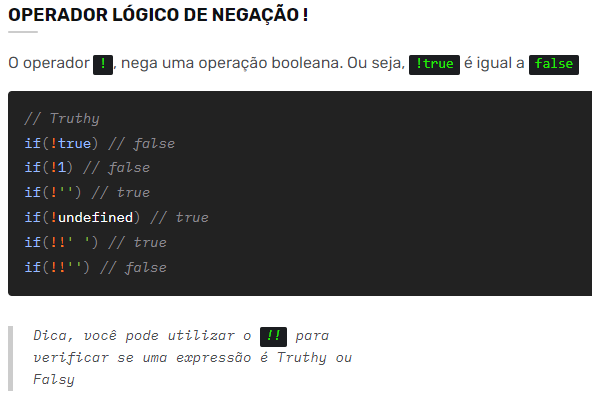
if(' ')

if('andre')

if(-5)

if({})

**1.4.e Operador logico de negação**



**1.4.f Operadores de comparação**

Retorna o valor booleano

**Operadores de maior 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)

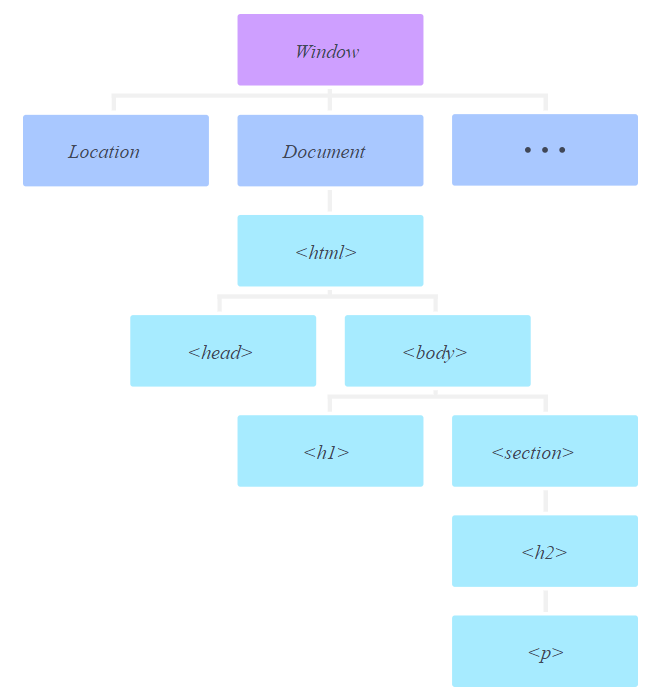
*const* nome = 'carlos' *// impossivel alterar o valor de uma const*

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))

**3.2.i – Arrow Function Reduzido**

*const* aulas = [

  {

    nome: 'HTML 1',

    min: 15

  },

  {

    nome: 'HTML 2',

    min: 10

  },

  {

    nome: 'CSS 1',

    min: 20

  },

  {

    nome: 'JS 1',

    min: 25

  },

]

*// caso queira retornar apenas um valor,*

*// cada aula(item) retorna aula.min*

tempoAulas = aulas.map(*aula* => aula.min) *// falando com cada aula > o valor de aula.min*

console.log(tempoAulas) *// [15, 10, 20, 25]*

**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**

Necessita 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*

**5.0 Objetos**

**5.1 Constructor Function – Função Construtora**

*// A função construtora tem a inicial maiuscula*

*// só compoe o objeto que será criado os itens que são adicionados com o this*

*function* Carro2(*modeloAtribuido*, *precoInicial*, *anoAtribuido*) {

*const* taxa = 1.2; *// caso queira exibir a taxa, substitua const por this*

*const* precoFinal = precoInicial \* taxa

  this.modelo = modeloAtribuido;

  this.preco = precoFinal;

  this.ano = anoAtribuido;

  }

*// modeloAtribuido, precoInicial, anoAtribuido*

*const* honda = new Carro2('Civic', 1000, 2010);

*const* volkswagem = new Carro2('Golf', 2000, 2017);

*// console retorna ~ Carro2 {modelo: 'Golf', preco: 2400, ano: 2017}*

Exemplo 1

*//nessa função construtora vamos criar 4 pessoas, onde podemos utilizar o metodo pessoa1.andar() para executar a função*

*//função recebendo dois parametros*

*function* Pessoa(*nome*, *idade*) {

  this.nome = nome;

  this.idade =idade;

*//caso metodo seja ativado ira realizar a função*

  this.andar = *function*(){

    console.log(nome + ' andou');

  }

}

*// criação de pessoas*

*const* pessoa1 = new Pessoa('Julia', 18);

*const* pessoa2 = new Pessoa('João', 20);

*const* pessoa3 = new Pessoa('Maria', 25);

*const* pessoa4 = new Pessoa('Bruno', 15);

*//executando metodo*

console.log(pessoa1.andar())

Exemplo real de função construtora

*//Aqui criamos uma Constructor Function (Dom) para manipulação de elementos com QueryselectorAll*

*//Onde temos dois metodos capazes de incluir e remover classes*

*function* Dom2(*itens*) { *// Desclaração da constructor com parametro*

*const* element2 = document.querySelectorAll(itens)*// Query Selector All com o parametro da constructor*

  this.elements = element2 *// metodo que retorna a lista, nesse caso, ul.elements ou item.elements retorna toda a nodelist da const selecionada*

this.incluir = *function*(*nomeclasse*) {*// metodo com função, recebendo da declaração do metodo o nome da classe a ser inclusa*

  element2.forEach((*tipo*) => { *// para cada item da lista*

  tipo.classList.add(nomeclasse) *// executamos a adição da classe*

  })

}

this.remover = *function*(*nomeremove*){*// metodo com função, recebendo da declaração do metodo o nome da classe a ser inclusa*

  element2.forEach((*tipo*) => { *// para cada item da lista*

  tipo.classList.remove(nomeremove)*// executamos a remoção da classe*

  })

}

}

*const* item = new Dom2('li'); *//declaração da const com a tag do elemento a ser incluso na constructor, pode ser uma classe ou uma id*

item.incluir('ola') *// declarando o metodo, e enviando no parametro a classe a ser inclusa*

item.remover('oi1')*// declarando o metodo, e enviando no parametro a classe a ser removida*

*const* ul = new Dom2('ul');

ul.incluir('ola-ul')

*// Crie uma função que retorne novos elementos*

*// html, com os seguintes parâmetros*

*// tag, classe e conteudo.*

*function* Cria(*tag*, *classe*, *conteudo*) {

*const* elem = document.createElement(tag);

  classe ? elem.classList.add(classe) : null

  conteudo ? elem.innerText = conteudo : null

  return elem;

}

console.log(Cria("h1", "ativo", "olá"));

**5.2 Prototype**

Prototype é um objeto que consegue adicionar novas propriedades a array

Sendo assim, ao criar um novo prototype de array, toda array terá essa função;

*const* frutas2 = ['Uva', 'Maçã', 'Banana'];

Array.prototype.mostratThis = *function*() {

  console.log(this) *//['Uva', 'Maçã', 'Banana']*

}

*//função construtora comum*

*function* Pessoa(*nome*, *idade*) {

  this.nome = nome;

  this.idade = idade;

}

*//Prototype é um objeto que consegue adicionar novas propriedades a array*

*//Criando prototype criamos funções isoladas, que objetos novos vão ter acesso, mas nao na construtora*

*//nesse caso:*

*//estamos adicionando na função construtora Pessoa por via de prototype + function*

*//a propriedade andar, que nesse caso adiciona uma string a propriedade da Pessoa*

Pessoa.prototype.andar = *function*() {

  return this.nome + ' Andou'

}

*/\**

*andre.andar()*

*'Andre Pessoa Andou'*

*\*/*

*//Podemos adicionar varios como;*

Pessoa.prototype.nadar = *function*() {

  return this.nome + ' Nadou'

}

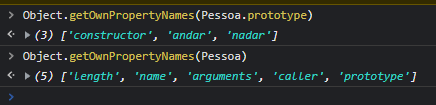
*/\**

*andre.nadar()*

*'Andre Nadou'*

*\*/*

*const* andre = new Pessoa('Andre', 23)



**5.2.a Entendendo o que está retornando;**

*const* Carro = { *// type off Carro = Object*

  marca: 'Ford', *// type off Carro.marca = String ou seja, aceita apenas metodos de string*

  preco: 2000,*// type off = Number // aceita metodos de Number*

*// executando função, nesse caso o que importa é o resultado true ou false, tem propriedades de valor booleano*

*// para acessar os metodos e propriedades dessa função executada é; Carro.andar.*

*// Carro.andar.constructor.name = Function, já quando ativa; Carro.andar().constructor.name é uma string*

  andar() {

    return true;

  }

}

Carro *// Object*

Carro.marca *// String // retorna a pripriedade string*

Carro.preco *// Number // retorna a propriedade que é um number*

Carro.acelerar *// Function // sem ativar a função, é uma function*

Carro.acelerar() *// Boolean // ativando a função se torna true ou false, que é string*

Carro.marca.charAt *// Function // sem ativar  = function*

Carro.marca.charAt(0) *// String // ativando, pega a letra que é uma string*

**5.2.b Verificando os metodos**

que podem ser utilizados com Nodelist HTML collection Document

Object.getOwnPropertyNames(NodeList.prototype)

Object.getOwnPropertyNames(Document.prototype)

Object.getOwnPropertyNames(HTMLCollection.prototype)

Podemos verificar o que cada propriedade é com o .constructor

*// se verifica por ex; li.click.constructor*

li; *// HTMLLIElement()*

li.click; *// Function()*

li.innerText; *//String()*

li.value; *//Number()*

li.hidden; *//Boolean()*

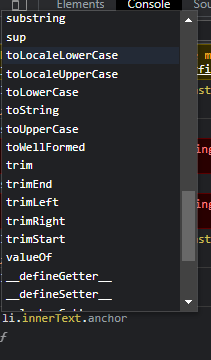
li.offsetLeft; *//Number()*

li.click(); *// não ativa nada*

*// Qual o construtor do dado abaixo:*

li.hidden.constructor.name; *//String()*

e verificamos os metodos e propriedades que podemos utilizar colocando por ex o item + innertext. E ver a listagem



**5.3 Objects - Native, host e user**

DIFERENTES VERSÕES

* Browsers diferentes

Apesar de tentarem ao máximo manter um padrão, browsers diferentes possuem objetos com propriedades e métodos diferentes.

* Versões de browsers

Sempre que o browser é atualizado, novos objetos, métodos e propriedades podem ser implementados.

* Host e Native Objects

Por exemplo, browsers que não implementaram o ECMAScript 2015 (ES6), não possuem o método find de Array.

**Nativos**

Objetos nativos são aqueles definidos na especificação da linguagem e são implementados independente do host.

*// Construtores de objetos nativos*

Object

String

Array

Function

**Host**

Objetos do host são aqueles implementados pelo próprio ambiente. Por exemplo no browser possuímos objetos do DOM, como DomList, HTMLCollection e outros. Em Node.js os objetos do Host são diferentes, já que não estamos em um ambiente do browser.

*// Objetos do browser*

NodeList

HTMLCollection

Element

**User**

USER

Objetos do user, são os objetos definidos pelo seu aplicativo. Ou seja, qualquer objeto que você criar ou que importar de alguma biblioteca externa.

*const* Pessoa = {

  nome: 'André';

}

**Verificar se existe**

VERIFICAR SE EXISTE

O typeof retorna o tipo de dado. Caso esse dado não exista ou não tenha sido definido, ele irá retornar undefined. Ou seja, quando não for undefined quer dizer que existe.

if (typeof Array.from !== "undefined") {

  console.log('existe')

} else {

  console.log('não existe')

}

if (typeof NodeList !== "undefined");

**5.4 Metodos e propriedades da STRING**

É a construtora de strings, toda string possui as propriedades e métodos do prototype de String.

*//aqui temos um objeto e uma string, mas ambas respondem aos metodos de string*

*const* comida = 'Pizza'; *// string*

*const* liquido = new String('Água'); *//  objeto*

*//comida.toLocaleUpperCase() = 'PIZZA'*

*// liquido.toLocaleUpperCase()  = 'ÁGUA'*

*const* comida2 = 'Pizza';

*const* frase = 'A melhor comida';

**5.4.a Length**

*// quantidade de caractere na string*

comida.length; *// 5*

frase.length; *// 15*

**5.4.b Charat**

*// Encontra qual a letra em determinada localização*

*//podemos acessar os caracteres da string cm se fosse uma array*

comida2[0] *// P // primeira letra da const comida2*

frase[0] *// A // primeira frase da const frase*

frase[frase.length - 1] *// a // descobre qual o compramento da string, e volta para a ultima letra, pois comça com zero*

*//podemos utilizar também com o comando charAt;*

console.log(frase.charAt(3))

console.log(frase.charAt(frase.length - 1))

**5.4.c Contatenando Strings**

*const* frase2 = 'A melhor linguagem é ';

*const* linguagem = 'JavaScript';

*//Podemos contaternar normalmente com string + string;*

*const* frasecompleta = frase2 + linguagem + '!!'

console.log(frasecompleta)

*//ou com o termo concat*

*const* fraseCompleta = frase2.concat(linguagem, '!!');

console.log(fraseCompleta)

**5.4.d Includes**

*//Procura uma string exata dentro de outra string*

*const* fruta = 'Banana';

*const* listaFrutas = 'Melancia, Banana, Laranja';

fruta.includes(listaFrutas); *// false // fruta não contem toda listaFrutas*

listaFrutas.includes(fruta); *// true // listaFrutas contem toda fruta*

**5.4.e Startswith Endswith**

*//verifica se a informação passada come a ou termina na string*

*const* fruta2 = 'Banana';

console.log(fruta.endsWith('nana')); *// true*

console.log(fruta.startsWith('Ba')); *// true*

console.log(fruta.startsWith('na')); *// false*

**5.4.f Slice**

*//corta a area indicada de uma string*

*const* transacao1 = 'Depósito de cliente';

*const* transacao2 = 'Depósito de fornecedor';

*const* transacao3 = 'Taxa de camisas';

*// cortando as 3 primeiras letras da string*

transacao1.slice(0, 3); *// Dep*

transacao2.slice(0, 3); *// Dep*

transacao3.slice(0, 3); *// Tax*

transacao1.slice(12); *// cliente // corta os 12 'primeiros' caracteres e considera somente o restante*

transacao1.slice(-4); *// ente // corta os 4 'ultimos' caracteres da string e considera somente o restante*

transacao1.slice(3, 6); *// ósi //  considera somente do terceiro ao sexto caractere*

**5.4.g Indexof e Lastindexof**

*//retorna a localização do primeiro caractere(ou grupo) que ele achar conforme solicitado*

*// no caso de last indexof retorna o ultimo*

*const* instrumento = 'Guitarra';

instrumento.indexOf('r'); *// 5*

instrumento.indexOf('ta'); *// 3*

instrumento.lastIndexOf('r'); *// 6*

**5.4.h Padstart**

*//deixa a string com a quantidade de espaço informada, e nesses espaços coloca o infomrado na declaração da propriedade*

*//adicionando traços no item 0*

*const* listaPrecos = ['R$ 99', 'R$ 199', 'R$ 12000'];

listaPrecos[0].padStart(20, '-') *//'---------------R$ 99'*

*// alinhando itens, dizendo que cada um deve ter 20 caracteres, e preencher com -*

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

  console.log(item.padStart(20, '-'))

})

*//---------------R$ 99*

*//--------------R$ 199*

*//------------R$ 12000*

**5.4.i Repeat**

*//repete a string a quantidade de vezes solicitada*

*const* frase3 = 'Ta ';

console.log(frase3.repeat(5)); *// TaTaTaTaTa*

**5.4.j Replace**

*// substitui o valor da const pelo passado, podemos passar direto ou com regular expression;*

*//Direto;*

*let* preco = 'R$ 1200,43';

preco = preco.replace(',', '.'); *// 'R$ 1200.43' // substituiu o primeiro , que encontrou por .*

*//regular expression \*\**

*//aqui precisamos substituir todos os espaços por ', '*

*let* listaItens = 'Camisas Bonés Calças Bermudas Vestidos Saias';

*// passamos o valor a ser substituido dentro das chaves, e após a virgula e entre aspas o que deve ficar no lugar*

listaItens = listaItens.replace(/[ ]+/g, ', ');

*//listaitens = 'Camisas, Bonés, Calças, Bermudas, Vestidos, Saias'*

**5.4.kSplit e Join**

*//transforma a string em array, quebrando pelos caracteres informados. e retirnado eles;*

*const* listaItens2 = 'Camisas Bonés Calças Bermudas Vestidos Saias';

*const* arrayItens = listaItens2.split(' '); *// quebrando quando houve espaço, caso não passar valor, quebra cada caractere em uma propriedade de string*

console.log(arrayItens)

*/\* resultado*

*0:"Camisas"*

*1:"Bonés"*

*2:"Calças"*

*3:"Bermudas"*

*4:"Vestidos"*

*5:"Saias"*

*\*/*

*const* htmlText = '<div>O melhor item</div><div>A melhor lista</div>';

*const* htmlArray = htmlText.split('div');

console.log(htmlArray)

*/\**

*0:"<"*

*1:">O melhor item</"*

*2:"><"*

*3:">A melhor lista</"*

*4:">"*

*\*/*

*// Join junta as propriedades da array em uma só string;*

*const* htmlNovo = htmlArray.join('section');

console.log(htmlNovo)

*//<section>O melhor item</section><section>A melhor lista</section>*

**5.4.l toUpperCase e toLowerCase**

*// Deixa a string em maiuscula ou minuscula*

*//como exemplo vamos receber digitando pelo usuario uma string*

*const* sexo1 = 'Feminino';

*const* sexo2 = 'feminino';

*const* sexo3 = 'FEMININO';

*//primeiro deixamos tudo em minusula, assim todas as opções são true na conferencia*

(sexo1.toLowerCase() === 'feminino'); *// true*

(sexo2.toLowerCase() === 'feminino'); *// true*

(sexo3.toLowerCase() === 'feminino'); *// true*

**5.4.m Trim**

*//Remove espaços em geral, começo ou fim*

*const* valor = '  R$ 23.00   '

valor.trim(); *// 'R$ 23.00'*

valor.trimStart(); *// 'R$ 23.00   '*

valor.trimEnd(); *// '  R$ 23.00'*

**5.5 Number**

É a construtora de números, todo número possui as propriedades e métodos do prototype de Number. Number também possui alguns métodos.

**5.5.a verificar se é numero isNaN**

Number.isNaN(NaN); *// true // is not a number? true*

Number.isNaN(4 + 5); *// false is not a number? false*

**5.5.b isInteger verifica se é integral**

Number.isInteger(20); *//  true // inteiro*

Number.isInteger(23.6); *// false // tem decimal*

**5.5.c ParseFloat**

Transforma string em number caso ela começe com numero

*// parseFloat transforma em number mesmo com decimal*

Number.parseFloat('99.50'); *// 99.5 // transforma em Number*

Number.parseFloat('100 Reais'); *// 100 // transforma em Number*

Number.parseFloat('R$ 100'); *// NaN // não transforma pois tem caractere na frente*

**5.5.d ParseInt**

Transforma em numero integral

*// 10 por padrão, significa casas decimais*

parseInt('99.50', 10); *// 99 // transforma em integral*

parseInt(5.43434355555, 10); *// 5*

Number.parseInt('100 Reais', 10); *// 100*

**5.5.e toFixed**

Arredonda com base no total de casas em argumento

*const* preco = 2.99;

preco.toFixed(); *// 3 // arredondou pra cima*

*const* carro = 1000.455;

carro.toFixed(2) *// 1000.46 // com argumento de invez de 3, duas casas decimais, arredondou pra baixo*

*const* preco2 = 1499.49;

preco2.toFixed() *// 1499 // arredondou ora baixo*

*const* preco3 = 10.32434;

console.log(preco3.toFixed()) *// 10*

**5.5.f toString**

Transforma number em String

*const* preco4 = 2.99;

preco.toString(10); *// '2.99'*

**5.5g LocalString**

Transforma um number em uma string de moeda indicada

*// a linguagem, seguido de um objeto com style e currency*

*const* reais = 59.60;

*const* stringReal = reais.toLocaleString('pt-BR', {style: 'currency', currency: 'BRL'})

console.log(stringReal)

**5.6 Math**

É um Objeto nativo que possui propriedades e métodos de expressões matemáticas comuns.

**5.6.a Arredondamento de números**

Math.abs(-5.5); *// 5.5 // Transforma em positivo*

Math.ceil(4.8334); *// 5 // Arredonda pra cima*

Math.ceil(4.3); *// 5 // Arredonda pra cima*

Math.floor(4.8334); *// 4 // arredonda pra baixo*

Math.floor(4.3); *// 4 // arredonda pra baixo*

Math.round(4.8334); *// 5 // arredonda pro mais proximo*

Math.round(4.3); *// 4 // arredonda pro mais proximo*

**5.6.b Min e max**

Math.max(5,3,10,42,2); *// 42 // retorna o maior*

Math.min(5,3,10,42,2); *// 2 // retorna o menor*

**5.6.c Gera numero aleatório**

*//gerar numero aleatorio*

Math.random(); *// 0.XXX*

Math.floor(Math.random() \* 100); *// entre 0 e 100*

console.log(Math.floor(Math.random() \* 100)) *// entre 0 e 100 arredondando para baixo,*

Math.floor(Math.random() \* 500);  *// entre 0 e 500*

*// Número random entre 72 e 32*

*// numero maximo > numero minumi +1 e + numero minimo*

Math.floor(Math.random() \* (72 - 32 + 1)) + 32;

*//Math.floor(Math.random() \* (max - min + 1)) + min;*

**5.6.d tres pontos para verificar um math**

... três pontos passa cada propriedade da string em um verificador

*// Retorne o maior número da lista abaixo*

*const* numeros0 = '4, 5, 20, 8, 9';

*const* numerosNum = numeros0.split(', ')

*const* numeroMax = Math.max(...numerosNum)

console.log(numeroMax)

**6.7 Array**

Arrays armazenam uma coleção de elementos. Estes podem ser strings, arrays, boolean, number, functions, objects e mais.

*//Arrays \*\**

*const* instrumentos = ['Guitarra', 'Baixo', 'Violão'];

*const* precos = [49, 99, 69, 89];

*const* dados = [new String('Tipo 1'),

['Carro', 'Portas', {cor: 'Azul', preco: 2000}],

*function* andar(*nome*) { console.log(nome) }];

dados[1][1]; *// acessando o segundo item, do segundo objeto*

dados[1][2]; *// azul // acessando o segundo item do terceiro objeto*

dados[1][2].cor; *// azul // acessando a propriedade do segundo item do terceiro objeto*

**6.7.a Construção de Arrays**

*//Toda array herda os métodos e propriedades do protótipo do construtor Array.*

*const* instrumentos2 = ['Guitarra', 'Baixo', 'Violão'];

*const* carros0 = new Array('Corola', 'Mustang', 'Honda');

carros[1] *// Mustang // retorna o segundo item*

carros[2] = 'Ferrari'; *// atribui valor ao terceiro item*

carros[10] = 'Parati'; *// caso atribua em uma casa que seja sepadara, ela criara uma array do tamanho, nesse caso 11, e deixara o restante vazio*

carros.length; *// 11 // tamanho da array*

**6.7.b Array From – transforma em array**

***//transforma um objeto que se parece com um array em array, como por ex*** *uma nodelist*

*//Nodelist para array*

*let* li = document.querySelectorAll('li'); *// Selecionando uma NodeList em uma variavel*

li = Array.from(li); *// Array // li recebe Array.from() com o nome da var em propriedade*

*// objeto para array*

*const* carros1 = {

  0: 'Fiat',

  1: 'Honda',

  2: 'Ford',

  length: 4, *// o objeto deve ter a declaração de comprimento com o lenght*

}

*const* carrosArray = Array.from(carros); *// variavel recebe Array.from() com o objeto em proriedade*

*// Selecione cada curso e retorne uma array*

*// com objetos contendo o título, descricao,*

*// aulas e horas de cada curso*

*const* cursos = document.querySelectorAll('.curso')

*const*  arrayCursos = Array.from(cursos)

*const* ObjetosCurso = arrayCursos.map((*curso*) => {

*const* titulo = curso.querySelector('h1').innerText;

*const* descricao = curso.querySelector('p').innerText;

*const* aulas = curso.querySelector('.aulas').innerText;

*const* horas = curso.querySelector('.horas').innerText;

  return {

    titulo,

    descricao,

    aulas,

    horas,

  }

})

**6.7.c IsArray – verifica se é array**

*//Utilizado o metodo*

Array.isArray(carrosArray)

true

Array.isArray(carros2)

false

**6.7.d Criação de array - Array.of Array() e New Array()**

Array.of(10); *// [10] // cria uma array com um item*

Array.of(1,2,3,4); *// [1,2,3,4] // cria uma array com a qnt de itens passados*

new Array(5); *// [,,,,] // cria uma array vazia com 5 espaços*

Array(5); *// [,,,,] // cria uma array vazia com 5 espaços*

Array(1,2,3,4); *// [1,2,3,4] // cria uma array com um item*

**6.7.e Length – tamanho da array**

*const* frutas = ['Banana', 'Pêra', ['Uva Roxa', 'Uva Verde']];

frutas.length; *// 3 // a array frutas tem 3 items (dois itens e uma array dentro da outra)*

frutas[0].length; *// 6 //retorna qmt de caractere do primeiro item*

frutas[1].length; *// 4 //retorna qmt de caractere do segundo item*

frutas[2].length; *// 2 // retorna a qnt de itens da array dentro da array*

frutas[2][0].length; *// 8 // retorna qnt de caractere do primeiro item da array dentro da array(item 2)*

**6.7.f Sort**

*// organiza a array*

*// Modifica a array para nova ordem*

*//no caso de string deixa em ordem alfabetoca*

*const* instrumentos3 = ['Guitarra', 'Baixo', 'Violão'];

instrumentos.sort();

instrumentos; *// ['Baixo', 'Guitarra', Violão]*

*//no caso de numeros, vai de caractere em caractere e deixa em ordem, não em ordem numerica*

*const* idades = [32,21,33,43,1,12,8];

idades.sort();

idades; *// [1, 12, 21, 32, 33, 43, 8]*

**6.7.g Unshift e Push – adicionando e removendo elementos**

*//aadicionar elementos no inicio ou fim da array, e retornar o length*

*//unshift - adiciona no começo*

*const* carros2 = ['Ford', 'Fiat', 'VW'];

carros.unshift('Honda', 'Kia'); *// 5 // retorna que tem 5 itens*

carros; *// ['Honda', 'Kia', 'Ford', 'Fiat', 'VW']; // var carros com os itens adicionados no começo*

*// push - adiciona no fim*

carros.push('Ferrari'); *// 6 //// retorna que tem 6 itens*

carros; *// ['Honda', 'Kia', 'Ford', 'Fiat', 'VW', 'Ferrari'];// var carros com o iten adicionado no fim*

**6.7.h Shift e Pop Remove itens**

*const* carros3 = ['Ford', 'Fiat', 'VW', 'Honda'];

*const* primeiroCarro = carros.shift(); *// 'Ford' // a variavel recebe o primeiro item da array, e tira o valor da array*

carros; *// ['Fiat', 'VW', 'Honda'];*

*const* ultimoCarro = carros.pop(); *// 'Honda' // a variavel recebe o ultimo item da array, e tira o valor da array*

carros; *// ['Fiat', 'VW'];*

**6.7.i Reverse – invertendo a ordem**

*//[].reverse() inverte os itens da array e retorna a nova array.*

*const* carros4 = ['Ford', 'Fiat', 'VW', 'Honda'];

carros.reverse(); *// ['Honda', 'VW', 'Fiat', 'Ford'];*

**6.7.j Splice adicionando em index**

*apartir do index passado, adiciona os elementos passados, e remove a qnt de elemento passado*

*const* carros5 = ['Ford', 'Fiat', 'VW', 'Honda'];

*// primeiro parametro = a partir / seundo parametro qnt a remover / e elementos a serem adicionados*

carros5.splice(1, 0, 'Kia', 'Mustang'); *// [] // // dps do primeiro item, adiciona e não remove*

carros; *// ['Ford', 'Kia', 'Mustang', 'Fiat', 'VW', 'Honda']*

carros5.splice(3, 2, 'Ferrari'); *// ['Fiat', 'VW'] // // dps do terceiro item, remove dois itens e adiciona ferrari*

carros5; *// ['Ford', 'Kia', 'Mustang', 'Ferrari', 'Honda']*

**6.7.k Copy within**

*[].copyWithin(alvo, inicio, final) a partir do alvo, ele irá copiar a array começando do inicio até o final e vai preencher a mesma com essa cópia. Caso omita os valores de início e final, ele irá utilizar como inicio o 0 e final o valor total da array.*

['Item1', 'Item2', 'Item3', 'Item4'].copyWithin(2, 0, 3); *// Seleciona ate terceiro item, e vai copiar do item zero ate o item 3*

*// ['Item1', 'Item2', 'Item1', 'Item2']*

['Item1', 'Item2', 'Item3', 'Item4'].copyWithin(-1);

*// ['Item1', 'Item2', 'Item3', 'Item1']*

**6.7.l Fill() – preencher**

*//Da valor os elementos da array*

['Item1', 'Item2', 'Item3', 'Item4'].fill('Banana'); *// preenche todos*

*// ['Banana', 'Banana', 'Banana', 'Banana']*

['Item1', 'Item2', 'Item3', 'Item4'].fill('Banana', 2); *// preenche a partit do item 2*

*// ['Item1', 'Item2', 'Banana', 'Banana']*

['Item1', 'Item2', 'Item3', 'Item4'].fill('Banana', 1, 3); *// preenche apartir  do item 1 ate o item 3*

*// ['Item1', 'Banana', 'Banana', 'Item4']*

**6.7.m Concat – Contatenar**

*//concatenar arrays e strings*

*const* transporte1 = ['Barco', 'Aviao'];

*const* transporte2 = ['Carro', 'Moto'];

*const* transportes = transporte1.concat(transporte2);

*// ['Barco', 'Aviao', 'Carro', 'Moto'];*

*const* maisTransportes = [].concat(transporte1, transporte2, 'Van');

*// ['Barco', 'Aviao', 'Carro', 'Moto', 'Van'];*

**6.7.n Inclues, Indexof e Lastindexof**

*//[].includes(valor) verifica se a array possui o valor e retorna true ou false.*

*//[].indexOf(valor) verifica se a array possui o valor e retorna o index do primeiro valor na array.*

*//[].lastIndexOf(valor) retorna o index do último.*

*const* linguagens = ['html', 'css', 'js', 'php', 'python', 'js'];

linguagens.includes('css'); *// true //sim, inclui*

linguagens.includes('ruby'); *// false //não inclui*

linguagens.indexOf('python'); *// 4 // index de python*

linguagens.indexOf('js'); *// 2 // index do primeiro js*

linguagens.lastIndexOf('js'); *// 5 // index do ultimo js*

**6.7.o Join - Juntar valores Split – Separar**

*//[].join(separador) junta todos os valores da array e retorna uma string com eles.*

*//Se você passar um valor como parâmetro, este será utilizado durante a junção de cada item da array.*

*// Join*

*const* linguagens2 = ['html', 'css', 'js', 'php', 'python'];

linguagens.join(); *// 'html,css,js,php,python'*

linguagens.join(' '); *// 'html css js php python'*

linguagens.join('-\_-'); *// 'html-\_-css-\_-js-\_-php-\_-python'*

**6.7.p Split separa**

*let* htmlString = '<h2>Título Principal</h2>'

htmlString = htmlString.split('h2');

*// ['<', '>Título Principal</', '>']*

htmlString = htmlString.join('h1');

*// <h1>Título Principal</h1>*

**6.7.q Slice Corta pedaço**

*//[].slice(inicio, final) retorna os itens da array começando pelo início e indo até o valor de final.*

*const* linguagens3 = ['html', 'css', 'js', 'php', 'python'];

linguagens.slice(3); *// ['php', 'python'] // corta os 3 primeiros*

linguagens.slice(1, 4); *// ['css', 'js', 'php'] // corta a partir do primeiro ate o quarto*

**6.7.r clonar array**

*const* original = carros.slice() *// recorta todos itens, e cola na nova const*

**6.8 Metodos de interação objetos Array**

**6.8.a Diferenças entre Array e MAP**

// Se o objetivo for uma ação de por exemplo, adicionar classe a elementos html, é utilizado o foreach

// caso o objetivo seja ter uma array com os dados modificados, criamos uma nova array com os dados modificados utilizando o método MAP invez de Foreach

**6.8.b ForEach**

*// foreach \*\**

*//a função de callback é executada para cada item da array,*

*//Ela possui três argumentos,*

*//item (valor do item da array),*

*//index (index do valor na array)*

*//array (array original).*

*const* carros = ['Ford', 'Fiat', 'Honda'];

carros.forEach(*function*(*item*, *index*, *array*) {

  console.log(item) *// qual item esta rodando a array*

  console.log(index) *// qual o index do item rodando*

  console.log(array) *// array que esta rodando*

  console.log(item.toUpperCase());

});

*// com Arrow Function*

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

  console.log(item.toUpperCase());

});

**6.8.c Modificando a array original com o foreach**

*//Modificando Array original com foreach  \*\**

*const* carros2 = ['Ford', 'Fiat', 'Honda'];

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

  array[index] = 'Carro ' + item; *//array recebe informação conforme index*

});

carros; *// ['Carro Ford', 'Carro Fiat', 'Carro Honda']*

**6.8.d Arrow Function**

É uma forma de abreviar a descrição do function com =>

*/Arrow Function \*\**

*const* li = document.querySelectorAll('li');

*//caso de só um argumento, nao precisa de chaves*

li.forEach(*i* => i.classList.add('ativa'));

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

  item.classList.add('ativa');

});

**6.8.e Map**

// Map - Cria nova array com dados modificados \*\*

//Metodo correto de retornar dados modificados um a um, como no foreach

//sempre utilizar esse metodo quando o intuito dor modificar os valores da array

*const* carros3 = ['Ford', 'Fiat', 'Honda'];

*const* novosCarros = carros3.map((*item*) =>{

  item += ' Novo'

  return item *// sempre retornar o item modificado*

})

*//novosCarros*

*//['Ford Novo', 'Fiat Novo', 'Honda Novo']*

*//ex retornando uma array com um elemento da outra array*

*const* aulas = [

  {

    nome: 'HTML 1',

    min: 15

  },

  {

    nome: 'HTML 2',

    min: 10

  },

  {

    nome: 'CSS 1',

    min: 20

  },

  {

    nome: 'JS 1',

    min: 25

  },

]

*// caso queira retornar apenas um valor,*

*// cada aula(item) retorna aula.min*

tempoAulas = aulas.map(*aula* => aula.min) *// falando com cada aula > o valor de aula.min*

console.log(tempoAulas) *// [15, 10, 20, 25]*

**6.8.f Criando uma const com uma função embutida**

*// realizando map, chamando uma const com função \*\**

*const* nomeAulasF = *function*(*aula*) { *// array recebendo função que recebe aula*

    return aula.nome*// e retorna aula.nome*

}

*// Também pode ser um arrow ;*

*const* nomeAulasF2 = *aula* => aula.nome

*const* arrayNomeAulas = aulas.map(nomeAulasF);  *// nova array realizando map com a const nomeAulasF que tem uma função*

**6.9 Reduce – Acumulador**

// Funciona da mesma forma que o map e o foreach

// Ele tem uma propriedade acumuladora, que é destacada entre as chaves da função

// esse valor acumula a cada rodada

// podendo realizar equações retornar o valor final

**6.9.a Reduce Equações**

*// Reduce Equações*

*const* aulas = [10,25,30];

*const* reduceAulas = aulas.reduce((*acumulador*, *item*, *index*, *array*) => { *// item adicional chamado aqui de "acumulador"*

  console.log(acumulador, item, index, array) *// apenas para acompanhamento,*

  return acumulador + item *// a cada rodada ele acumula o item + o numero predefinido, nesse caso 0*

}, 0) *// numero pre definido, // caso não seja passado nenhum valor, considerado 0 e a primeira interação não é realizada, apenas somado com as outras*

console.log (reduceAulas) *// 65 // soma total de todos os itens*

**6.9.b Reduce maior ou menor numero**

*//Reduce maior ou menor numero*

*const* numeros = [10,25,30,3,43,2];

*const* maiorNumero = numeros.reduce((*anterior*, *atual*) => { *// passado o item de acumulador como anterior e o item cm atual*

  if (anterior > atual)

    return anterior

  else

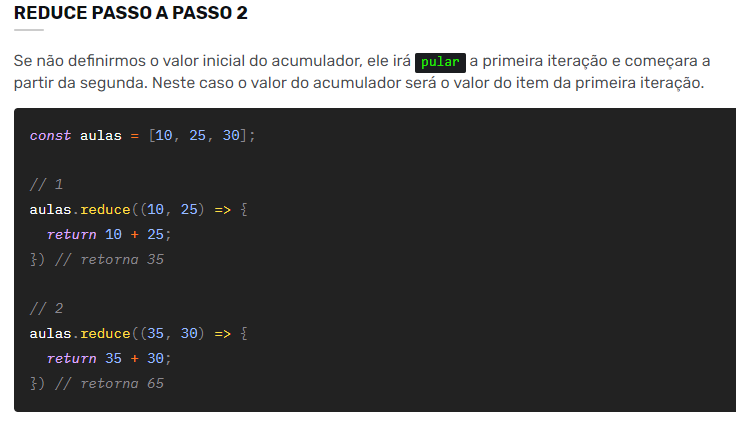
    return atual

}, 0) *// adicionado numero zero para acontecer desde o primeiro loop*

console.log(maiorNumero)

**6.9.c Reduce passo a passo**





**6.9.d Retornando propriedades de objetos com reduce**

*// Retornando propriedades de um objeto com o reduce \*\**

*const* aulas = [

  {

    nome: 'HTML 1',

    min: 15

  },

  {

    nome: 'HTML 2',

    min: 10

  },

  {

    nome: 'CSS 1',

    min: 20

  },

  {

    nome: 'JS 1',

    min: 25

  },

]

*const* listaAulas = aulas.reduce((*acumulador*, *atual*, *index*) => { *// parametros acumulador, que recebe o valor, atual, que é cada curso, e o index que é a posição*

  acumulador[index] = atual.nome; *// {0: 'HTML 1', 1: 'HTML 2', 2: 'CSS 1', 3: 'JS 1'} // acumlador[index] recebe o nome de cada curso(atual)*

  return acumulador; *// retorna o acumulador a listaAulas*

}, {}) *// invez de numeros, passamos chaves para acessar o objeto*

*// Retorne o valor total das compras*

*const* compras = [

  {

    item: 'Banana',

    preco: 'R$ 4,99'

  },

  {

    item: 'Ovo',

    preco: 'R$ 2,99'

  },

  {

    item: 'Carne',

    preco: 'R$ 25,49'

  },

  {

    item: 'Refrigerante',

    preco: 'R$ 5,35'

  },

  {

    item: 'Quejo',

    preco: 'R$ 10,60'

  }

]

*const* total = compras.reduce((*acumulador*, *item*, *index*) => {

*const* limpo = +item.preco.replace(',', '.').replace('R$ ', '')

  return acumulador + limpo

},0)

**6.9.e Reduce Right**

*// Inverte a ordem de retorno do reduce*

*const* frutas = ['Banana', 'Pêra', 'Uva'];

*const* frutasRight = frutas.reduceRight((*acc*, *fruta*) => acc + ' ' + fruta);

*const* frutasLeft = frutas.reduce((*acc*, *fruta*) => acc + ' ' + fruta);

frutasRight; *// Uva Pêra Banana*

frutasLeft; *// Banana Pêra Uva*

**7.0 Some**

// se pelo menos uma das interações for verdadeira, retorna true

*const* frutas2 = ['Banana', 'Pêra', 'Uva'];

*const* temUva = frutas.some((*item*) => { *// variavel que vai receber o valor booleano*

  return item == 'Uva' *// retorna true ou false dependendo se a variavel tiver o item especificado*

})

**7.1 Every**

// Se todas as interações forem verdadeiras, retorna true, caso uma seja false retorna false

*// verificando se todos os valores sao verdadeiros*

*const* every = frutas2.every((*item*) => { *// variavel que vai receber o valor booleano*

  return item  *// se dentro da array tiver qualquer valor como string vazia, null, undefined, retornara false*

})

*// verificando se todos os numeros são maiores que 3*

*const* numeros = [6, 43, 22, 88, 101, 29];

*const* maior3 = numeros.every((*item*) => {

  return item > 3 *// se maior que 3, recebe true*

})

**7.2 Find e Find Index**

**7.2.a Find**

//[].find(), retorna o valor atual da primeira iteração que retornar um valor truthy.

*const* numeros2 = [6, 43, 22, 88, 101, 29];

*const* buscaMaior45 = numeros2.find(*x* => x > 45); *// 88 // se a afirmação for verdadeira, retorna o conteudo da string*

**7.2.b FindIndex**

//ao invés de retornar o conteudo, retorna o index que encontrou

*const* buscaUva = frutas2.findIndex((*item*) => {

  return item === 'Uva' *// 3 // retorna o index que a string uva se encontra*

})

**7.3 Filter**

// Já o filter retorna uma array, então faz uma busca completa

*// Buscando apenas valores verdadeiros em uma array*

*const* frutas3 = ['Banana', undefined, null, '', 'Uva', 0, 'Maçã'];

*const* arrayFrutas = frutas3.filter((*item*) => {

  return item *// (3) ['Banana', 'Uva', 'Maçã']*

})

*//Buscando numerosmaiores*

*const* numeros4 = [6, 43, 22, 88, 101, 29];

*const* buscaMaior = numeros.filter(*x* => x > 45); *// [88, 101]*

**7.3.a Filter em Objetos**

*// Filter em objetos \*\**

*const* aulas2 = [

  {

    nome: 'HTML 1',

    min: 15

  },

  {

    nome: 'HTML 2',

    min: 10

  },

  {

    nome: 'CSS 1',

    min: 20

  },

  {

    nome: 'JS 1',

    min: 25

  },

]

*const* aulasMaiores15 = aulas2.filter((*aula*) => {

  return aula.min > 15; *// acessando objeto, retorna o objeto*

});

*// [{nome: 'CSS 1', min: 20}, {nome: 'JS 1', min: 25}]*

7.4 Function

7.4.a Propriedades

// Podemos utilizar as propriedades como length em uma função

//Caso a função resulte em uma string, tera propriedades de string, caso numero Number

*function* somar(*n1*, *n2*) {

  return n1 + n2;

}

console.log(somar.length) *// 2 argumentos*

console.log(somar.name) *// 'somar' // nome da funcao*

7.4.b Call

Call é uma função que ajuda a alterar o contexto da função chamada.

Em termos leigos, ajuda a substituir o valor this dentro de uma função

por qualquer valor que você desejar.

*//dois objetos*

*const* carro = {

  marca: 'Ford',

  ano: 2018

}

*const* carro2 = {

  marca: 'Vw',

  ano: 2019

}

*// função que recebe as propriedades dos objetos*

*function* descricaoCarro() {

  console.log(this.marca + ' Ano ' + this.ano);

}

descricaoCarro.call(carro) *// Ford Ano 2018*

descricaoCarro.call(carro2) *// Vw Ano 2019*

// This em Call \*\*

//O valor de this faz referência ao objeto criado durante a construção do objeto (Constructor Function).

//Podemos trocar a referência do método ao this, utilizando o call().

Ao chamar uma Call, o que importa é o parâmetro passado depois do call;

*const* carros = ['Ford', 'Fiat', 'VW'];

*// foreach normalmente*

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

  console.log(item);

}); *// Log de cada Carro*

*//foreeach com call*

carros.forEach.call(carros, (*item*) => {

  console.log(item);

}); *// Log de cada Carro*

*// podemos utilizar a foreach criada para carros para realizar com frutas*

*// ao realizar a call o que importa é o parametro passado depois da call*

*const* frutas = ['Banana', 'Pêra', 'Uva'];

carros.forEach.call(frutas, (*item*) => {

  console.log(item);

}); *// Log de cada Fruta*

*// Exemplo real*

*// 1- criando seletor de elemento*

*function* Dom(*seletor*) {

  this.element = document.querySelector(seletor); *// "seletor e element é o argumento"*

};

*// 2- criando aplicador de classe para o seletor*

Dom.prototype.ativo = *function*(*classe*) {

  this.element.classList.add(classe);

};

*// 3- selecionando o ul com o seletor dom*

*const* lista = new Dom('ul');

*// 4 - o selecionado chama o aplicador com o valor ativar*

lista.ativo('ativar'); *// ul recebe classe ativar*

console.log(lista);

*// 5 - selecionando um elemento em um objeto*

*const* novoSeletor = {

  element: document.querySelector('li') *// essencial a propriedade ter o mesmo nome, nesse caso 'element' para que possa utilizar o seletor*

}

*//  6- aplicador com call recebendo novo objeto e a classe ativar*

Dom.prototype.ativo.call(novoSeletor, 'ativar'); *// li recebe ativar*

*//O novo valor de this deve ser semelhante a estrutura do valor do this original do método.*

*//Caso contrário o método não conseguirá interagir de forma correta com o novo this.*

// ARRAY'S E CALL \*\*

//colocando a propriedade de array em um htmlcollection sem transformar em array

*// const li recebendo uma lista de elemento (nodelist)*

*const* li = document.querySelectorAll('li');

*//const filtro // Array.prototype com a propriedade filter com call referenciando a const li*

*const* filtro = Array.prototype.filter.call(li, (*item*) => {

*//retornando os itens que tenha class na classlist*

  return item.classList.contains('ativo')

})

console.log(filtro)

*// Retorne a soma total de caracteres dos*

*// parágrafos acima utilizando reduce*

*const* sec = document.querySelectorAll("p");

console.log(sec);

*const* conta = Array.prototype.reduce.call(sec, (*acc*,*item*) => {

  return (acc += item.innerText.length);

},0)

console.log(conta)

7.4.c Apply

//O apply(this, [arg1, arg2, ...]) funciona como o call, a única diferença é que

// os argumentos da função são passados através de uma array.

*const* numeros = [3,4,6,1,34,44,32];

Math.max.apply(null, numeros);

Math.max.call(null, 3, 4, 5, 6, 7, 20);

*// Podemos passar null para o valor*

*// de this, caso a função não utilize*

*// o objeto principal para funcionar*

7.4.d Bind

//Diferente de call e apply, bind(this, arg1, arg2, ...) não irá executar a função mas sim

//retornar a mesma com o novo contexto de this.

*// função construtora normal, com uma function*

*const* carro3 = {

  marca: 'Ford',

  ano: 2018,

  acelerar: *function*(*aceleracao*, *tempo*) {

    return **`**${this.marca} acelerou ${aceleracao} em ${tempo}**`**;

  }

}

*//função funciona normalmente com a function*

carro3.acelerar(100, 20);*// Ford acelerou 100 em 20*

*//criando um novo objeto*

*const* honda = {

  marca: 'Honda',

  ano: 2019,

};

*// com o metodo bind puxamos a função de dentro do construtor, e passamos os argumentos*

*// criando const que utiliza a construtora e a função, mas com o bind no objeto honda*

*const* acelerarHonda = carro3.acelerar.bind(honda); *// caso tenha um argumento fixo, colocar junto a declaração da cont; bind(honda, 100)*

acelerarHonda(200, 10);*// Honda acelerou 200 em 10*

*// Crie uma função que retorne novos elementos*

*// html, com os seguintes parâmetros*

*// tag, classe e conteudo.*

*function* Cria(*tag*, *classe*, *conteudo*) {

*const* elem = document.createElement(tag);

  classe ? elem.classList.add(classe) : null

  conteudo ? elem.innerText = conteudo : null

  return elem;

}

console.log(Cria("h1", "ativo", "olá"));

*// Crie uma nova função utilizando a anterior como base*

*// essa nova função deverá sempre criar h1 com a*

*// classe titulo. Porém o parâmetro conteudo continuará dinâmico*

*const* Cria2 = Cria.bind(null,'h1','titulo')

console.log(Cria2('olá'))

7.5 Object

Todo objeto é criado com o construtor Object e por isso herda as propriedades e métodos do seu prototype.

Segue duas formas de criar um objeto

*const* carro = {

  marca: 'Ford',

  ano: 2018,

}

*const* pessoa = new Object({

  nome: 'André',

  idade: 28,

})

7.5.a Métodos de Object - Objetos base com tipo prototipos

*//Object.create(obj, defineProperties) retorna um novo objeto*

*//que terá como protótipo o objeto do primeiro argumento.*

*//Objetos base com tipo prototipos\*\**

*// Objeto base com tipo prototipos para enviar aos novos objetos*

*//mas não entra em prototipos, entra dentro do objeto*

*const* carro = {

  rodas: 4,

  marca:'marca', *// caso o objeto que sera criado não tiver "marca", será utilizado esse*

  ano(*valor*) { *// forma de incluir um valor junto a criação do objeto*

    this.ano = valor

    return this*// retornando this objeto*

  },

  acelerar() {

    return this.marca + ' Acelerou'

  },

  buzinar() {

    return this.marca + ' Buzinou'

  },

}

*// criando objeto com base no objeto carro, esse objeto herda as funções*

*const* honda = Object.create(carro).ano(1995) *//.ano(1995) utilizando a função ano*

honda.marca = "Honda"

7.5.b Object Asign

*//Object.assign(alvo, obj1, obj2) adiciona ao alvo as propriedades e métodos*

*//enumeráveis dos demais objetos. O assign irá modificar o objeto alvo.*

*//Criação de um objeto com duas funções*

*const* funcaoAuto = {

  acelerar() {

    return 'Acelerou'

  },

  freiar() {

    return 'Freiou'

  }

}

*//Dois objetos*

*const* moto = {

  rodas: 2,

  capacete: true,

}

*const* carro2 = {

  rodas: 4,

  mala: true,

}

*//Colocando no prototype dos dois objetos as duas funções criadas*

Object.assign(carro2, funcaoAuto)

Object.assign(moto, funcaoAuto)

*//saida*

*//moto.acelerar()*

*//'Acelerou'*

*//moto.freiar()*

*//'Freiou'*

7.5.c DefineProperties

*// Define propriedade imutaveis*

*// criação de objeto vazio*

*const* motoR1 = {}

*// definindo as propriedades do objeto*

Object.defineProperties(motoR1, {

  rodas: {

    value: 2,

    configurable: false,*// impede deletar e mudança de valor*

    enumerable: true, *// torna enumerável*

  },

  capacete: {

    value: true,

    configurable: true,

    writable: false, *// impede mudança de valor*

  }

})

*const* quadrado = {}

Object.defineProperties(quadrado, {

  lados: {

    value: 10,

    configurable: false, *//padrao*

    writable: false, *//padrao*

    enumerable: true,

  }

})

7.5.d Get e Set

*// função em um objeto dentro de um objeto para setar valores*

*//função comum*

*const* bike = {

  capacete: true,

}

*// Defininfo propriedades*

Object.defineProperties(bike, {

  rodas: { *// objeto dentro do objeto*

    get() { *// pega o valor da const \_rodas*

      return this.\_rodas; *// retorna \_rodas*

    },

    set(*valor*) { *//seta o valor a \_rodas*

      this.\_rodas = valor \* 4 + " Total rodas" *// equação para dar valor a \_rodas com base no parametro passado*

    }

  }

})

*// bike.rodas //bike.rodas sem valor rpe definido = // undefined*

*// bike.rodas = 5 definindo o valor*

*// bike*

*// {capacete: true, \_rodas: 20} // bike já com o valor passado pela equação*

7.5.e GetOwnPropertyDescriptoys

*//Lista todos os métodos e propriedades de um objeto, com as suas devidas configurações.*

Object.getOwnPropertyDescriptors(Array);

*// Lista com métodos e propriedades e Array*

Object.getOwnPropertyDescriptors(Array.prototype);

*// Lista com métodos e propriedades do protótipo de Array*

Object.getOwnPropertyDescriptor(window, 'innerHeight');

*// Puxa de uma única propriedade*

7.5.f Object keys - values – entries

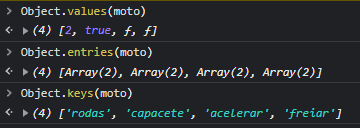
*//metodo para criar arrays com os objetos, esse metodo puxa apenas os enumerados*

*//Object.keys(obj) retorna uma array com as chaves de todas as propriedades diretas e enumeráveis do objeto.*

*//Object.values(obj) retorna uma array com os valores do objeto.*

*//Object.entries(obj) retorna uma array com array's contendo a chave e o valor.*

*const* motoR6 =  Object.keys(moto) *//motoR6 (4) ['rodas', 'capacete', 'acelerar', 'freiar']*



7.5.g Object GetOwnPropertyNames

*//metodo para criar arrays com os objetos, esse metodo sim puxa os enumerados*

Object.getOwnPropertyNames(Array);

*// ['length', 'name', 'prototype', 'isArray', 'from', 'of']*

Object.getOwnPropertyNames(Array.prototype);

*// [..., 'filter', 'map', 'every', 'some', 'reduce', ...]*

*const* carro5 = {

  marca: 'Ford',

  ano: 2018,

}

Object.getOwnPropertyNames(carro5);

*// ['marca', 'ano']*

7.5.h Object GetPrototypeOf e ObjectIs

*/Object.getPrototypeOf(), retorna o protótipo do objeto*

*//Object.is(obj1, obj2) verifica se os objetos são iguais e retorna true ou false.const frutas = ['Banana', 'Pêra']*

*const* frutas = ['Banana', 'Pêra']

Object.getPrototypeOf(frutas); *// retorna prototype de array*

Object.getPrototypeOf(''); *// retorna prototype de String*

*// Object.is(obj1, obj2) verifica se os objetos são iguais e retorna true ou false.*

*const* frutas1 = ['Banana', 'Pêra'];

*const* frutas2 = ['Banana', 'Pêra'];

Object.is(frutas1, frutas2); *// false // apesar de ter o mesmo conteudo, são objetos distintos*

*const* frutas3 = frutas1

console.log(Object.is(frutas1, frutas3)) *// True // verdadeiro pois frutas 3 é apenas referencia de frutas 1*

7.5.i Object freeze seal e prevenExtension

*// Object.freeze() impede qualquer mudança nas propriedades. Object.*

*// seal() previne a adição de novas propriedades e impede que as atuais sejam deletadas. Object.*

*// preventExtensions() previne a adição de novas propriedades.*

*const* carro6 = {

  marca: 'Ford',

  ano: 2018,

}

Object.freeze(carro6); *// não permite alterações*

Object.seal(carro6); *// sela o objeto, nao consegue incluir nem excluir*

Object.preventExtensions(carro6); *// não consegue mais adicionar e sim retirar*

*//verificação com object.is*

Object.isFrozen(carro6); *// true*

Object.isSealed(carro6); *// true*

Object.isExtensible(carro6); *// false*

*const* configuracao = {

  width: 800,

  height: 600,

  background: '#333'

}

Object.freeze(configuracao)

7.6 Propriedades e métodos do protótipo

Já que tudo em JavaScript é objeto, as propriedades abaixo estão disponíveis em todos os objetos criados a partir de funções construtoras. {}.constructor retorna a função construtora do objeto.

*const* frutas4 = ['Banana', 'Uva'];

frutas.constructor; *// Array //*

frutas.constructor.prototype *// prototipo do construtor array*

*const* frase = 'Isso é uma String';

frase.constructor; *// String*

frase.constructor.prototype *// construtor de string*

7.6.a asOwnProperty

*//verificar se a propriedade foi criada com ela*

*const* frutas5 = ['Banana', 'Uva'];

*//verificando se tem a propriedade*

frutas5.hasOwnProperty('map'); *// false // a propriedade não é do obj frutas*

Array.hasOwnProperty('map'); *// false // a propriedade não é da array*

Array.prototype.hasOwnProperty('map'); *// true // a propriedade é dp prototype de array*

*//O Objeto frutas5 tem somente as proriedades;*

*// Object.getOwnPropertyNames(frutas5)*

*// (3) ['0', '1', 'length']*

*// restante é herdado*

7.6.b PropertyIsEnumerable

*//{}.propertyIsEnumerable() verifica se a propriedade é enumerável.*

Array.prototype.propertyIsEnumerable('map'); *// false*

window.propertyIsEnumerable('innerHeight'); *// true*

7.6.c IsPrototyoeOf(Valor)

*//Verifica se é o protótipo do valor passado.*

*const* frutas6 = ['Banana', 'Uva'];

Array.prototype.isPrototypeOf(frutas6); *// true*

7.6.d To String

*// Retorna o tipo do objeto. O problema é toString() ser uma função dos protótipos de Array,*

*// String e mais. Por isso é comum utilizarmos a função direto do Object.prototype.toString.call(valor).*

*const* frutas = ['Banana', 'Uva'];

frutas.toString(); *// 'Banana,Uva'// transforma em String*

typeof frutas; *// object // informa que é um object*

Object.prototype.toString.call(frutas); *// [object Array] // informa que é um objeto, do tipo array*

*const* frase = 'Uma String';

frase.toString(); *// 'Uma String'*

typeof frase; *// string*

Object.prototype.toString.call(frase); *// [object String]*

*const* carro = {marca: 'Ford'};

carro.toString(); *// [object Object]*

typeof carro; *// object*

Object.prototype.toString.call(carro); *// [object Object]*

*const* li = document.querySelectorAll('li');

typeof li; *// object*

Object.prototype.toString.call(li); *// [object NodeList]*

*function* verificador(*obj*) {

  return Object.prototype.toString.call(obj)

}

console.log(verificador(frutas))

8.0 Efeitos no DOM

Todo elemento HTML do DOM herda propriedades e métodos do construtor HTMLElement.

*const* h1 = document.querySelector('h1');

Object.prototype.toString.call(h1); *// [object HTMLHeadingElement] // objeto do HTMLHeadingElement*

*// HTMLHeadingElement > HTMLElement > Element > Node > EventTarget > Object*

8.0.a Dataset

//ao adicionar um elemento no dom com a tag data- , o JS recebe esse elemento como objeto,

//com as informações como propriedades;

//<div data-cor="azul" data-width="500">Uma Div</div>

// Sempre que for interagir com um elemento por JS, utilizar o data-

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

console.log(div.dataset) *// DOMStringMap {cor: 'azul', width: '500'}*

*const* div2 = document.querySelector('[data-cor]')

console.log(div2) *// selecionando pelo nome da tag ou pelo data- seleciona o mesmo elemento*

*// podemos adicionar dataset por js*

div.dataset.height = 1000;

*//ou deletar*

delete div.dataset.width

//Transformação CamelCase

// Ao adicionar palavras contendo - após o data- o Js ao processar retira o traço e deica CamelCase;

//data-anima-scroll="left" = animaScroll: 'left'

//O mesmo acontece caso adicionar pelo Js com CamelCase, no HTML estará com –

*// Exemplo real*

*//Podemos adicionar data- a diferentes seções para animações diferentes*

<section data-anime="show-right">

<section data-anime="show-down">

// antes de adicionar a classe ativo,

//dicionamos a uma const a tab com o index e o data- que adicionamos as sessões

// e ao adicionar a classe ativo, adicionamos a const com o data- daquela sessão

const direcao = tabContent[index].dataset.anime;

      tabContent[index].classList.add("ativo", direcao);

//  dai criamos o css da classe ativa;

  .js-tabcontent section.ativo.show-right {

    display: block !important;

    animation: show-right .5s forwards;

  }

  .js-tabcontent section.ativo.show-down {

    display: block !important;

    animation: show-down .5s forwards;

  }

// e as animações para ocorrer diferentes

  @keyframes show-right {

    from {

      opacity: 0;

      transform: translate3d(-30px, 0, 0);

    }

    to {

      opacity: 1;

      transform: translate3d(0, 0, 0);

    }

  }

  @keyframes show-down {

    from {

      opacity: 0;

      transform: translate3d(0, -30px, 0);

    }

    to {

      opacity: 1;

      transform: translate3d(0, 0, 0);

    }

  }

8.1 Modules

Criamos cada interação de script em arquivos separados, e juntamos tudo utilizando um arquivo modulo;

Arquivo que importa os outros arquivos no html

  <script type="module" src="script/scriptModulado.js"></script>

*// importação deve começar com import + nome da função e from de onde ela vem em relação a este arquivo*

import initScrollSuave from './Modules/scrollSuave.js';

import initAccordion from './Modules/Accordion.js';

import initTabNav from "./Modules/TabNav.js"

import {teste1, teste2} from "./Modules/testes.js" *// importando duas funções no mesmo js*

*// Iniciando as importações*

initScrollSuave()

initTabNav()

initAccordion()

No arquivo de exportação devemos colocar export default antes da função

export default *function* initAccordion() {

## CARACTERÍSTICAS

* Strict mode

'use strict' por padrão em todos os arquivos.

* Variáveis ficam no module apenas

Não vazam para o escopo globo.

* This fora de um objeto faz referência a undefined

Ao invés de fazer referência ao window.

* Assíncrono

8.1.a Exportando valores

Podemos exportar objetos, funções, classes, números, strings e mais.

*// arquivo configuracao.js*

export *function* scrollSuave() {};

export *const* ano = 2000;

export *const* obj = {nome: 'Ford'};

export *const* str = 'Frase';

export *class* Carro {};

8.1.b use strict

O modo estrito previne que algumas ações consideradas erros. Basta adicionarmos 'use strict' no topo de um arquivo, que ele entrará neste modo.

*Por padrão todo module está no modo estrito*

'use strict';

nome = 'Ford'; *// erro, variável global*

delete Array.prototype; *// erro, não deletável*

window.top = 200; *// erro, não pode mudar*

*const* arguments = 3.14; *// escrever em palavra reservada*

8.2 Module

Janela suspensa ao clicar em um botão

Primeiro se deve criar o HTML para o o module após o footer, aqui por exemplo é um formulário de login, com um botão em X para fechar;

 <section class="modal-container ativo" data-modal="container">

      <div class="modal">

        <button data-modal="fechar" class="fechar">X</button>

        <form action="">

          <label for="email">Email</label>

          <input type="text" for="email" id="email" />

          <label for="senha">Email</label>

          <input type="password" for="senha" id="senha" />

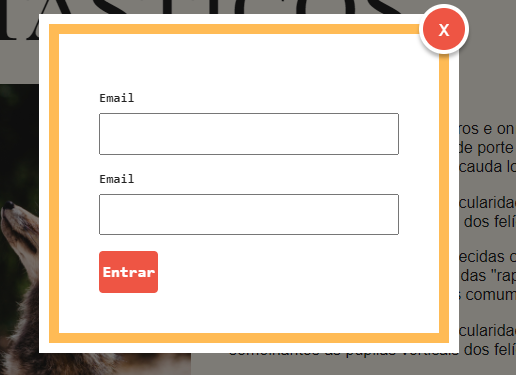
          <button type="submit">Entrar</button>

        </form>

      </div>

    </section>

Em seguida se deve criar o CSS para está área;



A area deve ter display none, e receber uma classe “ativo” com display flex

JS

Uma função toggle que verifica se o elemento da janela em si tem a classe ativo,

Caso não tiver adiciona ou remove

Outra função com parametro a area, e não o Modal, caso o clique for na area, ativa a função toggle

export default *function* initModal() {

*const* botaoAbrir = document.querySelector('[data-modal="abrir"]'); *// botao abrir*

*const* botaoFechar = document.querySelector('[data-modal="fechar"]'); *// botao fechar do modal*

*const* containerModal = document.querySelector('[data-modal="container"]'); *// conteiner do modal*

if (botaoAbrir && botaoFechar && containerModal) { *// se os 3 botoes existirem, executa*

*function* toggleModal(*event*) {

event.preventDefault() *//cm a pag ainda n existe prevent*

containerModal.classList.toggle('ativo') *// se não estiver com a classe adicina, se estiver remove*

}

*function* cliqueForaDoModal(*event*) {

  if (event.target === this) { *// se o target do click for no container do modal, e não especificamente no modal, executa o toggle*

    toggleModal(event)

  }

}

botaoAbrir.addEventListener('click', toggleModal)

botaoFechar.addEventListener('click', toggleModal)

containerModal.addEventListener('click', cliqueForaDoModal)

}

}

8.3 Animação ao Scroll

Toda vez que a section aparecer ela anima movimento lateral

Coloque uma classe data- em cada sessão desejada e as puxe no js

export default *function* animaScroll() {

*const* section = document.querySelectorAll('[data-anima="scroll"]')

if (section.length) {

*const* metadeWindow = window.innerHeight \* 0.6; *// const recebe o valor de 60% da janela do usuario*

*function* anima() {

    section.forEach((*section*) => { *// para cada sessão*

*const* secTop = section.getBoundingClientRect().top; *// const recebe o valor da distancia da sec para o top da tela*

*const* isSectionVisible = (secTop - metadeWindow) < 0; *// const verifica se  60% da tela menos a distancia do topo é menor que 0*

      if (isSectionVisible) { *// caso true ativa o IF*

        section.classList.add('ativo')

      }

    })

   }

anima() *// ativa a função ao menos uma vez, para o site iniciar visivel*

window.addEventListener('scroll', anima)

}

}

8.4 Tolltip

Caixa de informação que aparece sobre um item ao passar o mouse por cima

Adicione a área do html desejada o data-tooltip + aria-label

data-tooltip aria-label="">

O segundo passo é criar o JS até criar o element e então criar o CSS e voltar para o JS;

Para a criação do JS,

Selecionamos o elemento, adicionamos um evento de mouse over ativando uma função, essa função ativara a função de criação que retornara o elemento em uma const

Com essa const nos vamos adicionar o evento de mouse leave que levara para uma função que retornara a posição do mouse com um leve modificação + string com px

E também, ativara uma função de mouseleave que ira remover o element e os eventos criados

export default *function* initTooltip() {

}

*const* tooltips = document.querySelectorAll('[data-tooltip'); *// 1- Selecionamos o item que irá receber a tooltip*

tooltips.forEach((*item*) => { *// 2- caso tenha varios itens com tooltip, pada cada  um vamos adicionar um evenet listener*

  item.addEventListener('mouseover', onMouseOver ) *// 3- adicionando o event listener de mouse over, ativando a função onMouseOver*

})

*function* onMouseOver() { *// 3- função que ativa as outras*

*const* tooltipBox = criarToolTipBox(this);  *// 4- criação do elemento*

  this.addEventListener('mousemove', onMouseMove) *// 6 ativação da posição do mouse no mouse move*

  this.addEventListener('mouseleave', onMouseLeave) *// 9- ativa o removedor no mouse leave*

*function* onMouseMove(*event*) { *// 7- o elemento recebe a posição exata do mouse*

    tooltipBox.style.top = event.pageY + 20 + 'px'; *// 8- a posição exata do move é enviada com +20 px para se diatanciar do ponteiro*

    tooltipBox.style.left = event.pageX + 20 + 'px';*// 8- a posição exata do move é enviada com +20 px para se diatanciar do ponteiro*

  }

*function* onMouseLeave() { *// 10 - função que remove o element e os eventos*

    tooltipBox.remove() *//remove o elemento tooltip na saida do mouse*

    tooltipBox.removeEventListener('mouseleave', onMouseLeave)*// remove o evento*

    tooltipBox.removeEventListener('mousemove', onMouseMove)*//remove o evento*

}

}

*function* criarToolTipBox(*element*) { *// 5- Criação do elemento HTML*

*const* tooltipBox = document.createElement('div') *// a const recebe um create elemento com a tag selecionada*

*const* text = element.getAttribute('aria-label') *// const recebe o valor da string em aria-label*

  tooltipBox.classList.add('tooltip') *// adiciona a classe tooltip*

  tooltipBox.innerText = text *// o elemento div criado recebe o innertext do aria-label*

  console.log(tooltipBox) *// elemento div ja criado, com classe e texto*

*// adiciona a div ao final do body*

  document.body.appendChild(tooltipBox)

  return tooltipBox

*// se deve agora criar o CSS referente a esta tooltip*

}

CSS;

.tooltip {

  position: absolute;

  z-index: 500;

  background: rgba(0,0,0,.6);

  box-shadow: 0 0 0 2px #fb5,0 0 0 6px rgba(0,0,0,.6) ;

  padding: .5rem;

  color: white;

  font-family: monospace;

  font-weight: bold;

  font-size: .875rem;

  max-width: 150px;

  border-radius: 2px;

  top: 0px;

  left: 0px;

}

8.5 Objetos como funções

Utilizando objetos como call-back

Deveremos utilizar o método handle event, e dentro do handle, as propriedades do objeto devem ter o this.

Para envio do valor para o objeto;

  onMouseMove.tooltipBox = tooltipBox *// nome do objeto + nome da propriedade = nome do valor a enviar*

  this.addEventListener('mousemove', onMouseMove) *//this(objeto a enviar) adicionando evento de mousemove e ativando o bjeto que tem o handle*

E dentro do objeto vamos nos referir a propriedade com o this;

this.tooltipBox.style.top

Segue exemplo completo;

export default *function* initTooltip() {

*const* tooltips = document.querySelectorAll('[data-tooltip'); *// 1- Selecionamos o item que irá receber a tooltip*

tooltips.forEach((*item*) => { *// 2- caso tenha varios itens com tooltip, pada cada  um vamos adicionar um evenet listener*

  item.addEventListener('mouseover', onMouseOver ) *// 3- adicionando o event listener de mouse over, ativando a função onMouseOver*

})

*function* onMouseOver(*event*) {*// 4- Função onMouseOver*

*const* tooltipBox = criarToolTipBox(this);  *// 5- const tooltioBox recebe uma função que cria o elemento HTML*

  tooltipBox.style.top = event.pageY + 'px'; *// 7- CSS da tooltip referente a posição recebe onde o mouse entrou no elemento*

  tooltipBox.style.left = event.pageX + 'px';*// 7- CSS da tooltip referente a posição recebe onde o mouse entrou no elemento*

*// aqui temos dois objetos no callback, poderiam ser funções dentro dessa mesma função*

  onMouseMove.tooltipBox = tooltipBox *// 8- const que sera criada onMouseMove recebe tooltipbox como propriedade*

  this.addEventListener('mousemove', onMouseMove) *// 9- ao evento mousemove acionamos o objeto onMouseMove*

  onMouseLeave.tooltipBox = tooltipBox *//13- objeto onmouseleave rebe o elemento*

  onMouseLeave.element = this *//14 element de onMouseLeave recebe elemento*

  this.addEventListener('mouseleave', onMouseLeave) *// 15 adiciona o onMouseleave*

}

*const* onMouseMove = { *// 10- criação de objeto com função*

  handleEvent(*event*) { *//11- necessariamente deve ter o handle event para funcionar*

    this.tooltipBox.style.top = event.pageY + 20 + 'px'; *//12- a posição exata do move é enviada com +20 px para se diatanciar do ponteiro*

    this.tooltipBox.style.left = event.pageX + 20 + 'px';*//12- a posição exata do move é enviada com +20 px para se diatanciar do ponteiro*

  }

}

*const* onMouseLeave = {

  handleEvent() {*//16 como é um objeto devese declarar handle event*

    this.tooltipBox.remove() *//remove a tooltip na saida do mouse*

    this.element.removeEventListener('mouseleave', onMouseLeave)*// remove o evento*

    this.element.removeEventListener('mousemove', onMouseMove)*//remove o evento*

  }

}

*function* criarToolTipBox(*element*) { *// 6- Criação do elemento HTML*

*const* tooltipBox = document.createElement('div') *// a const recebe um create elemento com a tag selecionada*

*const* text = element.getAttribute('aria-label') *// const recebe o valor da string em aria-label*

  tooltipBox.classList.add('tooltip') *// adiciona a classe tooltip*

  tooltipBox.innerText = text *// o elemento div criado recebe o innertext do aria-label*

  console.log(tooltipBox) *// elemento div ja criado, com classe e texto*

*// adiciona a div ao final do body*

  document.body.appendChild(tooltipBox)

  return tooltipBox

*// se deve agora criar o CSS referente a esta tooltip*

}

}