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

**7.4 Function**

7.4.a Propriedades

7.4.b Call

7.4.c Apply

7.4.d Bind

**7.5 Object**

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

7.5.b Object Asign

7.5.c DefineProperties

7.5.d Get e Set

7.5.e GetOwnPropertyDescriptoys

7.5.f Object keys - values – entries

7.5.g Object GetOwnPropertyNames

7.5.h Object GetPrototypeOf e Objects

7.5.i Object freeze seal e prevenExtension

**7.6 Propriedades e métodos do protótipo**

7.6.a asOwnProperty

7.6.b PropertyIsEnumerable

7.6.c IsPrototyoeOf(Valor)

7.6.d To String

**8.0 Efeitos no DOM**

**8.0.a Dataset**

**8.1 Modules**

8.1.a Exportando valores

8.1.b use strict

**8.2 Janela suspensa**

**8.3 Animação ao Scroll**

**8.4 Tolltip**

**8.5 Objetos como funções**

**8.6 Dowp down menu**

**8.7 SetTimeOut**

8.7.a Metodo com contagem regressiva;

8.7.b SetTimeOut pertence ao window

8.7.c Set interval

8.7.d clear interval

8.7.e Ex. de interval

**8.8 Menu mobile**

**8.9 – observador**

**9.0 Animação em números**

**9.1 Data Object**

9.1.a Tempo até

9.1.b Horario de funcionamento

**9.2 Formularios**

9.2.a Values

9.2.b Validação

9.2.c Select Checkbox e Radio

9.2.d Recebendo os dados em um objeto

**9.3 Button Creator**

**9.4 SetValues**

**10.0 JavaScript Assíncrono**

10.1.a Promises – Then

10.1.b Promises – Catch

10.1.c Promise Finally

10.1.d Promise all

10.1.e Promise Race

**10.2 Fetch API**

10.2.a Json e then

10.2.b Blob

10.2.c Clone

10.2.d Headers

10.2.e Status

10.2.f URL

10.2.g Type

10.2.h Exercicios fetch

**10.3 Jason 2**

10.3.a Criando Array com objetos

10.3.b Metodo Parse para transformar text em objeto

10.3.c Transformando objeto em arquivo json

10.3.d Enviando objeto para o local storage como json

10.3.e voltando como objeto utilizando o parse

**10.4 API**

**10.5 HTTP**

**10.6 URL E METHOD**

10.6.a GET

10.6.b POST

10.6.cPUT

10.6.d HEAD

10.6.e Cors

**10.7 Async Await**

10.7.a Then e Async

10.7.b Try e Catch em async

10.7.c iniciando fetch ao mesmo tempo

10.7.d o resultando a frente do await deve ser uma promisse, caso não seja o await não tem efeito

**10.8 History API**

10.8.a PushState

10.8.b PopState

10.8.c FETCH E HISTORY

**10.9 Trocando Html da pag atual**

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)

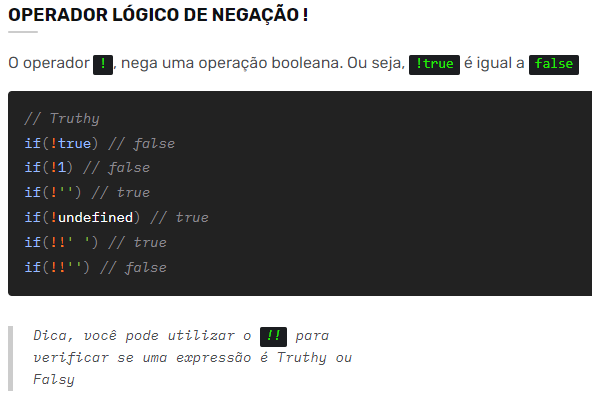
if(' ')

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 esta 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*) { // Adicionando função a uma propriedade

    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)

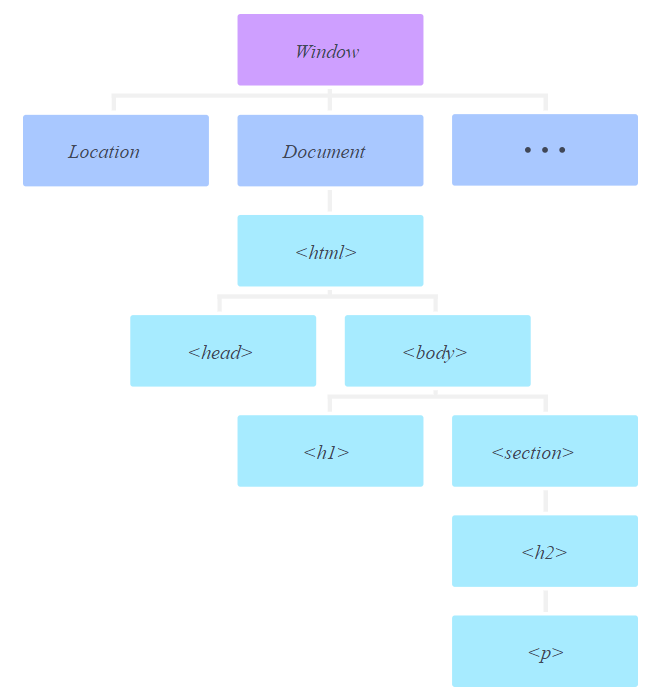
*const* nome = 'carlos' *// impossivel alterar o valor de uma const*

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**

Caso retorne apenas um valor, podemos reduzir o Arrow Function

*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 a classe*

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**

Combinando medias a medidas de janela

*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**

PreviousElementSibling = element anterior  
PreviousSibling = Qualquer item anterior

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 uma propriedade que todas as funções tem, e podemos utilizar o nome da função + prototype + nova função para criar novos métodos para essa função

Assim nas funções construtoras, todos os itens criados vão ter esse método novo

*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'

}

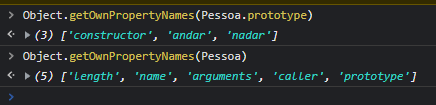
*/\**

*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()*

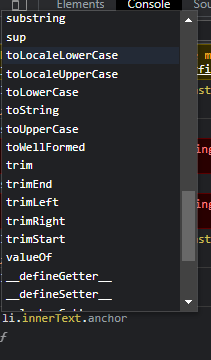
li.offsetLeft; *//Number()*

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 começa 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 informado 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

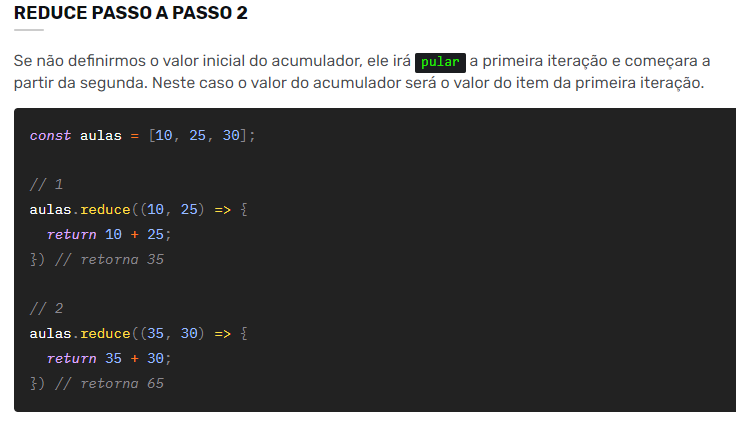
    return atual

}, 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.

Exemplo acionando uma função passando dois objetos diferentes

*//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 pre definido = // undefined*

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

*// bike*

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

**7.5.e GetOwnPropertyDescriptors**

*//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**

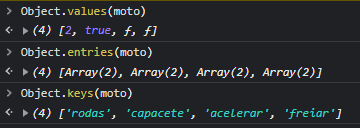
*//metodo para criar arrays com os objetos, esse metodo puxa apenas os enumerados*

*//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 Objects**

*/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 deixa 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,

//adicionamos 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 Janela Suspensa**

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" />

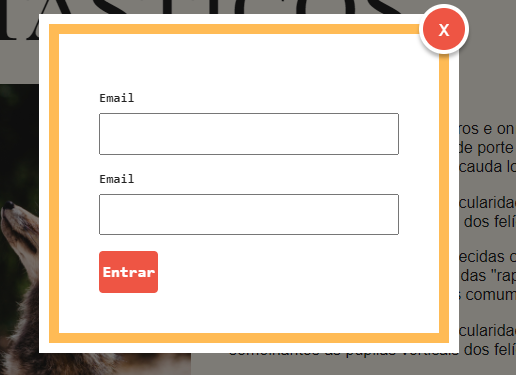
          <button type="submit">Entrar</button>

        </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*

}

}

**8.6 Dowp down menu**

Ao passar o mouse sobre uma opção, é exibido um dropdown com diversas opções

Primeiro passo

No menu com listas, se deve adicionar uma lista ul dentro da li desejada, adicionando as classes para edição de css e JS

      <nav class="menu" data-tab="menu"> // navegação do menu

        <ul> // lista do menu

          <li><a href="#animais">Animais</a></li> // opção do menu

          <li data-dropdown=""> // opção do menu com dropdown

            <a href="sobre.html">Sobre</a> // texto e link da opção

            <ul class="dropDownMenu"> // ul do dropmenu

                <li><a href="empresa.html">Empresa</a></li>

                <li><a href="equipe.html">Equipe</a></li>

                <li><a href="Socios.html">Socios</a></li>

                <li><a href="investidores.html">Investidores</a></li>

            </ul>

          </li>

          <li><a href="#faq">Faq</a></li>

          <li><a href="#contato">Contato</a></li>

          <li><a data-modal="abrir" href="login.html">Login -></a></li>

        </ul>

      </nav>

Segue o CSS para tal janela;

[data-dropdown] {

  position: relative; */\* li geral tem posição relative, para o ul responder a ele\*/*

}

[data-dropdown] .dropDownMenu { */\* display none quando não esta com hover \*/*

  display: none;

}

[data-dropdown]*:hover* .dropDownMenu, */\* em caso da classe ter hover ou a classe click \*/*

[data-dropdown].active .dropDownMenu { */\* ao hover na li a ul  pega display block com animação \*/*

  display: block;

  animation: show-up .2s forwards; */\*Chamada da animação\*/*

}

*/\*Animação da janela\*/*

@keyframes show-up {

  from {

    opacity: 0;

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

  }

  to {

    opacity: 1;

    transform: translate3d(0,0,0);

  }

}

.dropDownMenu { */\* Estilização da caixa ul \*/*

 position: absolute; */\* posição absoluta em relaçao a li \*/*

 top: 34px; */\* esta caixa deve estar perfeitamente grudada a li para que o hover funcione \*/*

 background-color: #e54;

 border: 2px solid black;

 z-index: 200;

}

.dropDownMenu*::before* { */\* criação de uma seta antes da ul \*/*

  content: '';

  display: block;

  position: absolute;

  top: -10px;

  width: 0px;

  height: 0px;

  border-left: 10px solid transparent;

  border-right: 10px solid transparent;

  border-bottom: 10px solid #e54;

}

.dropDownMenu li a*:hover*{ */\* ao hover no link \*/*

color: white;

}

Agora o CSS que irá adicionar e remover a classe da li e verificar o próximo click

import outsideclick from './outsideclick.js';

export default *function* dropDownMenu() {

*const* dropdown = document.querySelectorAll('[data-dropdown]'); *// 1- seleciona os itens do li*

*// 2-  para cada item da ul dentro da li, adiciona o evento que ao userEvent(trouch e click), aciona a function HandleClick*

dropdown.forEach(*menu* => {

  ['touchstart', 'click'].forEach(*userEvent* => {

    menu.addEventListener(userEvent, handleClick)

  })

});

*// 3-*

*function* handleClick(*event*) { *// 3- função que adiciona e remove a classe ao li*

  event.preventDefault(); *// previnimos o padão do link*

  this.classList.add('active') *// 4- adiciona a classe 'ativo' a li*

  outsideclick(this,  ['touchstart', 'click'], () => {  *// 5- chamamos a function outsideclick com parametro this, de click e retornando a callback se positivo*

    this.classList.remove('active') *// 18- remove a classe ativo caso retorne o callback*

  })

}

}

export default *function* outsideClick(*element*, *events*,  *callback*) { *// 6- recebendo o parametro this  // parametro de click // e a função do callback*

*const* html = document.documentElement *// 7- const recebe o html todo*

*const* outside = 'data-outside' *// 8- verifica se o click foi na caixa*

  if (!element.hasAttribute(outside)) { *// 9- se o element(this=li) não tiver o data-outside*

    events.forEach(*userEvent* => { *// 10- para cada item do events(click e touch)*

*//setTimeOut é adicionado, porque sem ele é realizado todo o processo já no primeiro clique, sendo assim quando clicamos no icone, que não é a li*

*// já está removido o atributo, colocando o set time out só é ativado a função de remoção no clique fora da caixa*

setTimeout(() =>  html.addEventListener(userEvent, handleClickOuside)) *// 11- adiciona ao html o event listener de click e touch acionando a function handleclickoutside,*

    })

  element.setAttribute(outside,'') *// 12- adiciona ao element(this=li) o data-outside*

  }

*function* handleClickOuside(*event*) {  *// 13- function que verifica se o click não foi na caixa da li*

    if(!element.contains(event.target)) { *// 14-  event.target é exatamente onde foi clicado, verifica se o click não foi em event (this=li)*

      element.removeAttribute(outside) *// 15- caso não tenha sido na caixam remove o atributo data-outside*

      events.forEach(*userEvent* => { *// 16- remove o event de click do html*

      html.removeEventListener(userEvent, handleClickOuside)

      })

    callback()*// 17- caso afirmações acima verdadeiras, adicona o callback da outsideclick*

    }

  }

}

**8.7 SetTimeOut**

Metodo que aciona um call-back após um tempo

O set time out é executado spós todo o código JS ter sido executado

Exemplos de uso;

*function* espera(*texto*) { *// function com log*

  console.log(texto);

}

setTimeout(espera, 1000, 'olá mundo'); *// metodo com a função, tempo e texto //1s*

setTimeout(*function*() { *// metodo já com função passando tempo após fechar chaves*

  console.log('testando')

}, 2000) *//2s*

setTimeout(() => { *// com arrow function*

  console.log('testando')

}, 3000) *//3s*

}

**8.7.a Metodo com contagem regressiva;**

for(*let* i = 0; i < 11; i++) {

  setTimeout(() =>{

    console.log(i);

  }, 1000 \* i) *// 1s \* o aumento de i em cada rodada*

}

**8.7.b SetTimeOut pertence ao window**

Sendo assim, ao utilizar para por ex incluir uma classe, exibira erro

  setTimeout(*function*() {

    this.classList.add('active');*// Erro pois window.classList não existe*

  }, 1000)

Então se desejar utilizar, utilizar com arrow function, pois puxa o pai da function

*const* btn = document.querySelector('button');

btn.addEventListener('click', handleClick);

*function* handleClick() {

  setTimeout(() => {

    this.classList.add('active');*// Erro pois window.classList não existe*

  }, 1000)

}

**8.7.c Set interval**

setInterval(callback, tempo, arg1, arg2, ...), irá ativar o callback toda vez que a quantidade de tempo passar.

*function* loop(*texto*) {

  console.log(texto);

}

setInterval(loop, 1000, 'Passou 1s'); *// a cada 1s exibe a msg desejada*

*// loop a cada segundo*

*let* i = 0;

setInterval(() => {

  console.log(**`**Pasou ${i++} Segundos**`**); *// exibe continuamente o avabço de i*

}, 1000);

**8.7.d clear interval**

*let* i = 0;

*const* meuLoop = setInterval(() => { *// o set deve estar em uma const*

  console.log(**`**Pasou ${i++} Segundos**`**); *// exibe continuamente o avabço de i*

  if (i > 10) {

    clearInterval(meuLoop) *//interrompe o loop // a const deve estar declarada no clear*

  }

}, 1000);

**8.7.e Ex. de interval**

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

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

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

iniciar.addEventListener('click', inicia)

pausar.addEventListener('click', pausa)

pausar.addEventListener('dblclick', reseta)

*let* i = 0; *// contador*

*let* timer; *// irá receber o setInterval*

*function* inicia() {

timer = setInterval(() => { *// inicia contador*

  tempo.innerText = i++

},100)

iniciar.setAttribute('disabled', '') *// desativa ao clicar*

}

*function* pausa() {

console.log('pausar')

clearInterval(timer) *// pausa o contador timer*

iniciar.removeAttribute('disabled', '') *// remove atributo de iniciar ao clicar*

 }

*function* reseta() { *// limpa contador*

  tempo.innerText = 0;

  i = 0

console.log('reseta')

 }

**8.8 Menu mobile**

Primeiro preciso criar o botao menu e si no html;

<button data-menu="button" aria-expanded="false" aria-controls="menu">Menu</button>

Seguido da ul do menu

 <ul data-menu="lista" id="menu">

E então refazer o css da barra de menu, para o tamanho que sera expandida

[data-menu="button"] {

  display: none; */\* não será exibito o botão em telas pequenas \*/*

}

@media (max-width:700px) {

  .js [data-menu="button"] { */\* personalização do menu \*/* */\* só ocorre caso tenha js. \*/*

    display: flex;

    background-color: #222;

    border: none;

    padding: .5rem 1rem;

    border-radius: 4px;

    text-transform: uppercase;

    font-family: monospace;

    font-weight: bold;

    color: white;

    cursor: pointer;

    margin: 10px;

    align-items: center;

    position: relative;

    z-index: 300;

  }

  .js [data-menu="button"]*::before* { */\* desenho hamburguer menu \*/*

    content: '';

    margin-right: 4px;

    display: inline-block;

    height: 2px;

    width: 12px;

    background-color: white;

    box-shadow: 0 4px white, 0 -4px#fff;

    transition: transform .3s;

  }

  .js [data-menu="button"].active*::before* { */\* ao ser ativo, o botão gera uma animação \*/*

    transform: rotate(90deg);

  }

  .js [data-menu="lista"] { */\* personalização do novo formato da lista de menu \*/*

    display: none;

    position: absolute;

    background-color: #fb5;

    width: 120px;

    z-index: 200;

    top: 0;

    padding: 60px 0 20px 0;

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

  }

  .js [data-menu="lista"].active { */\* animação a ser executada \*/*

    display: block;

    animation: show-right .5s forwards;

  }

    .js .dropDownMenu { */\* posição do drop menu dentro do menu mobile \*/*

    position: static;

    border: none;

  }

}

E então o JS

import outsideclick from './outsideclick.js';

export default *function* dropDownMenu() {

*const* dropdown = document.querySelectorAll('[data-dropdown]'); *// 1- seleciona os itens do li*

*// 2-  para cada item da ul dentro da li, adiciona o evento que ao userEvent(trouch e click), aciona a function HandleClick*

dropdown.forEach(*menu* => {

  ['touchstart', 'click'].forEach(*userEvent* => {

    menu.addEventListener(userEvent, handleClick)

  })

});

*// 3-*

*function* handleClick(*event*) { *// 3- função que adiciona e remove a classe ao li*

  event.preventDefault(); *// previnimos o padão do link*

  this.classList.add('active') *// 4- adiciona a classe 'ativo' a li*

  outsideclick(this,  ['touchstart', 'click'], () => {  *// 5- chamamos a function outsideclick com parametro this, de click e retornando a callback se positivo*

    this.classList.remove('active') *// 18- remove a classe ativo caso retorne o callback*

  })

}

}

export default *function* outsideClick(*element*, *events*,  *callback*) { *// 6- recebendo o parametro this  // parametro de click // e a função do callback*

*const* html = document.documentElement *// 7- const recebe o html todo*

*const* outside = 'data-outside' *// 8- verifica se o click foi na caixa*

  if (!element.hasAttribute(outside)) { *// 9- se o element(this=li) não tiver o data-outside*

    events.forEach(*userEvent* => { *// 10- para cada item do events(click e touch)*

*//setTimeOut é adicionado, porque sem ele é realizado todo o processo já no primeiro clique, sendo assim quando clicamos no icone, que não é a li*

*// já está removido o atributo, colocando o set time out só é ativado a função de remoção no clique fora da caixa*

       setTimeout(() =>  html.addEventListener(userEvent, handleClickOuside)) *// 11- adiciona ao html o event listener de click e touch acionando a function handleclickoutside,*

    })

  element.setAttribute(outside,'') *// 12- adiciona ao element(this=li) o data-outside*

  }

*function* handleClickOuside(*event*) {  *// 13- function que verifica se o click não foi na caixa da li*

    if(!element.contains(event.target)) { *// 14-  event.target é exatamente onde foi clicado, verifica se o click não foi em event (this=li)*

      element.removeAttribute(outside) *// 15- caso não tenha sido na caixam remove o atributo data-outside*

      events.forEach(*userEvent* => { *// 16- remove o event de click do html*

      html.removeEventListener(userEvent, handleClickOuside)

      })

    callback()*// 17- caso afirmações acima verdadeiras, adicona o callback da outsideclick*

    }

  }

}

**8.9 – observador**

Utilizado para verificar mudanças em objetos, como nesse caso por exemplo se houve animação por scroll

*// Observador \*\**

*// observa um objeto para verificar se houve alguma mudança*

*const* observeTarget = document.querySelector('.numeros') *// selecionando o numeros para identificar mudanças*

*const* observer = new MutationObserver(handleMutation); *// tipo de objeto que observa, acionando a function handlemutation*

observer.observe(observeTarget,{attributes: true})

*function* handleMutation(*mutation*) {

  if(mutation[0].target.classList.contains('ativo')) { *// verificando em mutation[0] se contem a classe ativo*

    observer.disconnect() *//caso sim, desconectar o observador*

    animaNumeros() *// e ai sim ativar a função desejada, nesse caso para dar inicio a contagem*

  }

}

**9.0 Animação em números**

Caso a pagina tenha animação por rolagem, é necessário colocar um observador para só ativar a função de contagem quando houver mudança na área desejada para contar

Para contar de 0 até um numero especifico, primeiro colocamos em uma array ou variável o numero com queryselector

Seguido de um for each que ira colocar em uma consto total, em outra o incremento da contagem e uma para o inicio da contagem

Uma const de timer recebe uma função anonima de setinterval que realiza a contagem na const start recebendo ela mesmoa + incremento

E o numero.innertext recebe o numero de start conforme ocorre a contagem

Seguido de um if para verificar se o numero passou o total, se sim o numero.innertext recebe o total, e é parado o contador com clear interval

O intervalo de contagem do setinterval é gerado aleatoriamente com \* math.random()

export default *function* initAnimaNumeros() {

*// Observador \*\**

*// observa um objeto para verificar se houve alguma mudança*

*const* observeTarget = document.querySelector('.numeros') *// selecionando o numeros para identificar mudanças*

*const* observer = new MutationObserver(handleMutation); *// tipo de objeto que observa, acionando a function handlemutation*

observer.observe(observeTarget,{attributes: true})

*function* handleMutation(*mutation*) {

  if(mutation[0].target.classList.contains('ativo')) { *// verificando em mutation[0] se contem a classe ativo*

    observer.disconnect() *//caso sim, desconectar*

    animaNumeros() *// e ai sim ativar a função desejada, nesse caso para dar inicio a contagem*

  }

}

*function* animaNumeros() { *// função de animação*

*const* numeros = document.querySelectorAll('[data-numero]') *// recebendo a array de nuemros*

numeros.forEach((*numero*) => { *// para cada item da array executa*

*const* total = +numero.innerText; *// colocando em total os itens da array transformados em numeros*

*const* incremento = Math.floor(total / 100) *// incremento que fara a contagem até o total, total / 100 e arredondado com math.floor*

*let* start = 0; *// inicio da contagem*

*const* timer = setInterval(() => { *// timer que realizara a contagem recebendo de uma função anonima de setInterval pora realizar a contagem*

    start = start + incremento; *// inicio da contagem recebe ele mesmo + o incremento*

    numero.innerText = start; *// a area de numero recebe o start que vai incrementando*

    if(start > total) { *// quando o start for maior que o igual*

    numero.innerText = total; *// a arrea numero recebe o total*

    clearInterval(timer) *// e o timer é parado*

    }

  }, 25 \* Math.random()) *// o intervalo de contagem de cada numero é gerado aleatoriamente*

})

}

}

**9.1 Data Object**

O construtor Date cria um objeto contendo valores como mês, dia, ano, horário e mais. A data é baseada no relógio interno do computador.

*const* agora = new Date

console.log(agora)

*// A partir do momento que uma const tem um new Date, ela se torna um objeto de date*

*// Por esse motivo tem prioridades como um objeto*

*// a contagem de dia da semana, ou mes do ano,começa a partir do 0 ex o domingo = 0*

*// Semana Mês Dia Ano HH:MM:SS GMT  hoje 13/04/23 14:08 quinta*

console.log(agora.getDate()) *// Dia // 13*

console.log(agora.getDay()) *// Dia da Semana ex: 5 = Fri // 4*

console.log(agora.getMonth()) *// Número dia mês // 3*

console.log(agora.getFullYear()) *// Ano // 2023*

console.log(agora.getHours()) *// Hora // 14*

console.log(agora.getMinutes()) *// Minutos // 8*

console.log(agora.getTime()) *// ms desde 1970 // 1681405863560*

console.log(agora.getUTCHours()) - 3 *// Brasília // 17*

**9.1.a Tempo até**

*// Dias até \*\**

*// tenha a data atual em uma const*

*const* atual = new Date

*// A data futura desejada deve ser passado para o objeto*

*const* futuro = new Date('dec 24 2023 23:59')

console.log(futuro) *// Sun Dec 24 2023 23:59:00 GMT-0300 (Horário Padrão de Brasília)*

*// A função de contagem deve ser feita com os milisegundos(getTime)*

*function* faltam(*tempo*) {

 return tempo / (24 \* 60 \* 60 \* 1000); *// 24 hrs \* 60min \* 60seg \* 1000ms*

}

*const* diaagora = faltam(atual.getTime())

*const* diafuturo = faltam(futuro.getTime())

*const* paraNatal = Math.floor(diafuturo - diaagora)

console.log(**`**Faltam ${paraNatal} Dias para o natal!**`**)

**9.1.b Horario de funcionamento**

Primeiro devemos declarar no local do html o data- referente aos dias da semana e o horário

          <li data-semana="1,2,3,4,5" data-horario="8,18">Seg à Sex das 8 as 18</li>

Em seguida CSS é criado o indicador primeiro em fechado, e caso estiver aberto, recebe a classe ativo

[data-semana]*::after* {

  display: inline-block;

  content: '';

  width: 6px;

  height: 6px;

  background: #e54;

  border-radius: 50%;

  margin-left: 4px;

  border: 2px solid black;

}

[data-semana].aberto*::after* {

  background: #3f4;

}

O JS selecionamos o data- e o transformamos em Array de numero utilizando split e map

Selecionamos a data e dia atual apartir de um new Date

E verificamos se o dia atual está dentro do funcionamento, e se o horário está entre o index 0 e 1

Enfim, caso o dia e o horário seja true, o objeto recebe a classe aberto

export default *function* initFuncionamento() {

*// Após adicionar no HTML os dias e o horario de funcionamento, puxamos ele para o JS*

*//deve-se criar o indicador fechado, e caso a classe for incluida, está indicandoaberto*

*const* funcionamento = document.querySelector('[data-semana]') *//  recebendo os dias da semana que funciona, como uma string*

*// Puxamos para outra const o data set com o devido nome*

*// Transformando em Array utilizando o split por virgula, e o map para transformar em numeros*

*const* diasSemana = funcionamento.dataset.semana.split(',').map(Number);

*const* funcionamento2 = document.querySelector('[data-horario')

*const* horario = funcionamento2.dataset.horario.split(',').map(Number)

*const* dataAgora = new Date() *// Utilizando o construtor na const*

*const* diaAgora = dataAgora.getDay() *// podemos puxar dela o dia atual*

*const* horarioAgora = dataAgora.getHours() *// e a hora atual*

*// recebe true ou false, se o index da semana for diferente de -1, recebe true // recebe menos 1 se for false porque não existe na array*

*const* semanaAberto = diasSemana.indexOf(diaAgora) !== -1;

*// se o horario estiver entre o index 0 e 1, recebe true ou false*

*const* horarioAberto = (horarioAgora >= horario[0] && horarioAgora < horario[1])

*// Se a const semanaAberto e horarioAberto for true, o objeto do data- selecionado recebe classe aberto*

if (semanaAberto && horarioAberto) {

  funcionamento.classList.add('aberto')

}

}

**9.2 Formularios**

É comum utilizarmos inputs de formulários para criarmos uma interface entre funções de JavaScript e o usuário final do site. Para isso precisamos aprender como pegar os valores dos formulários.

  <form name="contato" id="contato">

    <label for="nome">Nome</label>

    <input type="text" name="nome" id="nome">

    <label for="email">Email</label>

    <input type="email" name="email" id="email">

    <label for="mensagem">Mensagem</label>

    <textarea name="mensagem" id="mensagem"></textarea>

  </form>

console.log(document.forms); *// HTML collection dos formularios da pag*

console.log(document.forms.contato); *// Formulario de name contato*

console.log(document.forms.contato.elements); *// elementos do formulario contato*

console.log(document.forms[0].elements[0].value); *// valor do primeiro element do form. contato*

**9.2.a Values**

A propriedade value retorna o valor do elemento no formulário. Se adicionarmos um callback a por ex; keyup (tecla levantar), podemos ficar de olho no evento e puxar o valor sempre que ele mudar. change dispara quando houver mudanças.

Keyup = a cada tecla levantada, pega o valor

Change = a cada mudança de campo, pega o valor

*const* userName = document.querySelector('.userName')*// area que recebera o nome em change*

*const* contatoChange = document.querySelector('#nome') *// campo do nome do usuario*

*function* handleKeyUp(*event*) {

  userName.innerText = event.target.value *// função adiciona o valor de nome no campo userName*

}

contatoChange.addEventListener('change', handleKeyUp) *// função sendo ativada pelo change*

**9.2.b Validação**

Para efeito visual, e exibir mensagem caso o usuário não preencher um dado, pordemos colocar uma área após o input para exibir a mensagem, que pode ser utilizada para vários input;

    <span class="erro"></span>

    .invalido {

      border-color: red;

    }

*// Validação*

*const* verificacao = document.querySelector('#contato') *// selecionando toda lista de input*

*function* handleChange(*event*) {

*const* target = event.target

  if (!target.checkValidity()) { *// caso não foi valido*

    target.classList.add('invalido') *// é adicionado a classe de invalido*

    verificacao.nome.setCustomValidity(' Seu nome é importante!') *// mensagem de retorno personalizada para nome*

    target.nextElementSibling.innerText = target.validationMessage *// exibe mensagem de requisito padrão*

  }

}

verificacao.addEventListener('change', handleChange) *// ao change, xecuta a função de verificação*

**9.2.c Select Checkbox e Radio**

Select e input color

O valor será passado para o background de body, trocando o fundo da pagina

    <select name="cores" id="cores">

      <option value="">Selecione</option>

      <option value="blue">azul</option>

      <option value="red">vermelho</option>

    </select>

    <input type="color">

Checkbox

será passado true quando marcado

    <input type="checkbox" value="identidade" id="identidade">

Radio

Value será passado

    <label for="guitarra">guitarra</label>

    <input type="radio" id="guitarra" name="instrumento" value="guitarra">*<!-- name das ckechbox tem que ser igual -->*

    <label for="Baixo">Baixo</label>

    <input type="radio" id="Baixo" name="instrumento" value="baixo">

JS

*function* handleColor(*event*) {

  if(event.target.checked) { *// caso radio ou check estiver marcado;*

    console.log(event.target.checked)

    console.log(event.target.value)

  }

  document.body.style.backgroundColor = event.target.value *//style de body recebe o background com o value*

}

verificacao.addEventListener('change', handleColor)

**9.2.d Recebendo os dados em um objeto**

*//recebendo em objeto*

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

*const* dados = {} *// objeto que ira receber*

*function* handleChange(*event*) {

  dados[event.target.name] = event.target.value *// a propriedade name era o value preenchido em change*

}

contato.addEventListener('change', handleChange)

**9.3 Button Creator**

Para o site ter interação em tempo real com o que acorre no CSS

Utilizamos nomenclaturas no HTML de CSS substituindo o – por camelcase,

No CSS criamos uma function que contem uma const declarando o nome, e a outra o valor, então chamamos o objeto que tem todas as opções, enviando as duas const, cada propriedade dentro do objeto tem uma função que irá executar a mudança, e junto a chamada do objeto, executamos a show CSS se necessário

HTML;

  <section class="button-creator">

    <form id="controles">

      <label for="innerText">Texto</label>

      <input type="innerText" name="innerText" id="innerText">

      <label for="color">Color</label>

      <input type="color" name="color" id="color">

      <label for="backgroundColor">Background</label> *<!--Sempre com camelcase onde no CSS houver traço-->*

      <input type="color" name="backgroundColor" id="backgroundColor">

      <label for="height">Height</label>

      <input type="range" name="height" id="height" min="0" max="200">

      <label for="width">width</label>

      <input type="range" name="width" id="width" min="0" max="600">

      <label for="border">border</label>

      <input type="text" name="border" id="border">

      <label for="borderRadius">border-radius</label>

      <input type="range" name="borderRadius" id="borderRadius" min="0" max="20">

      <label for="fontFamily">Fonte</label>

      <select name="fontFamily" id="fontFamily">

        <option value="Arial">Arial</option>

        <option value="Georgia">Georgia</option>

        <option value="monospace">Monospace</option>

      </select>

      <label for="fontSize">Font-size</label>

      <input type="number" name="fontSize" id="fontSize">

    </form>

    <section class="resultado">

      <button class="btn">Clique</button>

      <pre><code class="css"></code></pre>

    </section>

  </section>

JS

*const* controles = document.querySelector('#controles') *// Selecionamos o Formulario*

*const* css = document.querySelector('.css') *// A area que exibira o CSS*

*const* btn = document.querySelector('.btn') *// o Botão que irá ser estilizado*

*function* handleChange(*event*) { *// função que cria a const, e chama o objeto e o show css*

*const* name = event.target.name *// const name guarda o nome do objeto clicado*

*const* value = event.target.value *// const valur, o valor*

  handleStyle[name](value); *// chamamos o objeto com o nome e o valor de cada*

  showCss() *// e após cada envio, executamos o show css*

}

controles.addEventListener('change', handleChange) *// a cada mudança, executamos o handle change, que irá executar o objeto*

*const* handleStyle = {*//objeto que contem em cada propriedade, uma função com o nome da operação em CSS*

  element: btn, *// elemento a ser mudado*

  innerText(*value*) {

    this.element.innerText =  value

  },

  color(*value*) {

    this.element.style.color = value

  },

  backgroundColor(*value*) {

    this.element.style.backgroundColor = value

  },

  height(*value*) {

    this.element.style.height = value + 'px'

  },

  width(*value*) {

    this.element.style.width = value + 'px'

  },

  border(*value*) {

    this.element.style.border = value + 'px'

  },

  borderRadius(*value*) {

    this.element.style.borderRadius = value + 'px'

  },

  fontFamily(*value*) {

    this.element.style.fontFamily = value

  },

  fontSize(*value*) {

    this.element.style.fontSize = value + 'rem'

  },

}

*function* showCss() { *// aqui separamos em block com span com split e join*

  css.innerHTML = '<span>' + btn.style.cssText.split('; ').join(';</span><span>');

}

**9.4 SetValues**

Temos a ferramenta de salvar opções do usuário para que ao atualizar a pagina não seja perdido

Essas opções são salvas como objetos com Keys e values

*function* handleChange(*event*) {

*const* name = event.target.name

*const* value = event.target.value

  handleStyle[name](value);

  saveValues(name, value); *// 1- colocamos a função de salvar para iniciar dentro da função de chamada*

  showCss()

}

*function* saveValues(*name*, *value*) {*// 2- função de salvar*

  localStorage[name] = value; *// salvará no objeto padrao localstorage, a propriedade com name e value*

}

*function* setValues() { *// 4- seta os valores salvos antes*

*const* properties = Object.keys(localStorage); *// a const properties recebe os valores do local storage*

  properties.forEach(*propriedade* => { *// e para cada valor*

    controles.elements[propriedade].value = localStorage[propriedade]; *//  const 'controles' que é o formulario, no indice a propriedade do looping, recebe o valor dessa propriedade do local storage*

    handleStyle[propriedade](localStorage[propriedade]); *// executa a função de setar valores com o local storage no indice do looping*

  })

  showCss(); *// executa também o show css*

}

*function* showCss() { *// aqui separamos em block com span com split e join*

  css.innerHTML = '<span>' + btn.style.cssText.split('; ').join(';</span><span>');

}

setValues() *// 3- inicia a função de setar os valores*

**10.0 JavaScript Assíncrono**

* Síncrono

Espera a tarefa acabar para continuar com a próxima.

* Assíncrono

Move para a próximo tarefa antes da anterior terminar. O trabalho será executado no 'fundo' e quando terminado, será colocado na fila (Task Queue).

* Exemplos

setTimeout, Ajax, Promises, Fetch, Async.

**10.1 Promisses**

Promise é uma função construtora de promessas.

Existem dois resultados possíveis de uma promessa,

ela pode ser resolvida, com a execução do primeiro argumento,

ou rejeitada se o segundo argumento for ativado.

*// Verifica se uma condição é verdadeira ou false e então resolve ou rejeita*

*// a Promisse é um objeto construtor*

*const* promessa = new Promise((*resolve*, *reject*) => {

*let* cond = true

if (cond) {

  setTimeout(() => { *// contador até 2s*

    resolve({nome: 'Jeziel', idade: 28}) *// Promise {<fulfilled>: 'Jeziel'}*

  }, 2000)

} else {

  reject('Falha') *// Promise {<rejected>: 'Falha'}*

}

})

**10.1.a Promises – Then**

O poder das Promises está no método then() do seu protótipo.

O Callback deste método só será ativado quando a promise for resolvida.

O argumento do callback será o valor passado na função resolve.\*/

*const* retorno = promessa.then((*resolucao*) => { *// quando a promessa for true, é executado*

  resolucao.profissao = 'WebDesigner' *// adiciona ao objeto mais uma propriedade*

  return resolucao *// retorna com a nova propriedade*

})

.then((*resolucao*) => { *// em seguida ja executa outro*

  console.log(resolucao)

  resolucao.hobby = 'Hotwheels'

  return resolucao

})

.then((*resolucao*) => {

  console.log(resolucao)

  resolucao.Sexo = 'Masc.'

  return resolucao

})

.then ((*resolucao*) => {

  console.log(resolucao)

})

console.log(retorno) *// {nome: 'Jeziel', idade: 28, profissao: 'WebDesigner', hobby: 'Hotwheels', Sexo: 'Masc.'}*

**10.1.b Promises – Catch**

O método catch(), do protótipo de Promises, adiciona um callback a promise que será ativado caso a mesma seja rejeitada.

.then ((*resolucao*) => {

  console.log(resolucao)

}).catch(*rejeitada* => {

  console.log(rejeitada)

})

.then ((*resolucao*) => {

  console.log(resolucao)

}, *rejeitada* => { *// podemos dar negação tambem como segundo argumento de then*

  console.log(rejeitada)

})

**10.1.c Promise Finally**

finally() executará a função anônima assim que a promessa acabar. A diferença do finally é que ele será executado independente do resultado, se for resolvida ou rejeitada.

.then ((*resolucao*) => {

  console.log(resolucao)

}, *rejeitada* => { *// podemos dar negação tambem como segundo argumento de then*

  console.log(rejeitada)

})

.finally(() => {

  console.log('Acabou')

})

**10.1.d Promise all**

Retornará uma nova promise assim que todas as promises dentro dela forem resolvidas ou pelo menos uma rejeitada. A reposta é uma array com as respostas de cada promise.

*const* login = new Promise((*resolve*) => {

  setTimeout(() => {

    resolve('Usuario Logado')

  }, 1000)

})

*const* dados = new Promise((*resolve*) => {

  setTimeout(() => {

    resolve('Usuario Carregados')

  }, 1500)

})

*const* carregouTudo = Promise.all([login, dados]) *// executa quando as duas promises forem executadas*

carregouTudo.then((*resolucao*) => {

  console.log(resolucao)

})

**10.1.e Promise Race**

Retornará uma nova promise assim que a primeira promise for resolvida ou rejeitada. Essa nova promise terá a resposta da primeira resolvida.

*const* login = new Promise((*resolve*) => {

  setTimeout(() => {

    resolve('Usuario Logado')

  }, 1000)

})

*const* dados = new Promise((*resolve*) => {

  setTimeout(() => {

    resolve('Usuario Carregados')

  }, 1500)

})

*const* carregouTudo = Promise.race([login, dados]) *// executa quando a primeira for concluida*

carregouTudo.then((*resolucao*) => {

  console.log(resolucao)

})

**10.2 Fetch API**

API significa que é uma interface de aplicação

*const* doc = fetch("./doc.txt"); *// realizando requisição com o metodo fetch*

doc.then((*resolucao*) => {*// utilizando then para interagir ccom o retorno de fetch, que é uma promise*

    return resolucao.text(); *// adicionando*

  })

  .then((*body*) => {*// repetindo o then com o metodo text() para interagir com a propriedade body*

    console.log(body);

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

conteudo.innerText = body

  });

**10.2.a Json e then**

Um tipo de formato de dados muito utilizado com JavaScript é o JSON (JavaScript Object Notation), pelo fato dele possuir basicamente a mesma sintaxe que a de um objeto js. .json() transforma um corpo em json em um objeto JavaScript.

*const* doc = fetch("./doc.txt"); *// realizando requisição com o metodo fetch*

doc.then((*resolucao*) => {*// utilizando then para interagir ccom o retorno de fetch, que é uma promise*

    return resolucao.text(); *// adicionando*

  })

  .then((*body*) => {*// repetindo o then com o metodo text() para interagir com a propriedade body*

    console.log(body);

*const* conteudo = document.querySelector('.conteudo') *// selecionando uma div do html*

    conteudo.innerText = body *// adicionando o texto do bloco de notas a div*

  });

*//jSON*

*// Selecionando um arquivo Json e transformando em um objeto*

*const* cep = fetch('https://viacep.com.br/ws/87114655/json/')

cep.then((*resolucao*) => {

  return resolucao.json();

})

.then((*cep*) => {

  console.log(cep);

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

  logadouro.innerText = cep.logradouro

});

*// Podemos também pegar todo um arquivo html e adicionar a uma div, ou partes dele*

*const* site = fetch('./site.html') *// puxando o arquivo html*

*const* div = document.querySelector('.html') *// selecionando a div que sera incluido*

site.then(*r* => r.text())

  .then(*body* => {

    console.log(body)

    div.innerHTML = body *// innerhtml da div recebe todo o body do arquivo html*

    console.log(div)

    console.log(div.querySelector('h1'))

  })

**10.2.b Blob**

Criando link de imagem e inserindo no DOM

*//Blob*

*// Inserção via fetch de imagem*

*const* img = fetch('./evolucao1.jpg'); *// seleciona a img na pasta*

  img.then(*r* => r.blob())

  .then(*body* => {

*const* blobURL = URL.createObjectURL(body) *// cria-se uma url da imagem e adiciona a uma const*

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

    imgDom.src = blobURL *// inserindo o link na url da tag img*

  })

**10.2.c Clone**

*//clone*

*// Para transformar tem a necessidade de clonar antes*

*const* cep = fetch('https://viacep.com.br/ws/87114655/json/'); *//*

  cep.then((*res*) => {

*const* res2 = res.clone() *// utilizando clone para colocar em uma const*

    res.text().then((*text*) => {  *// transformando em text*

      console.log(text) *// retorna texto*

    })

    res2.json().then((*json*) => { *// transformando em json*

      console.log(json) *// retorna objeto*

    })

  })

**10.2.d Headers**

Podemos dar console.log em headers para termos informações

*const* cep = fetch('https://viacep.com.br/ws/87114655/json/'); *//*

  cep.then((*response*) => {

    response.headers.forEach(console.log)

    });

**10.2.e Status**

*const* cep = fetch('https://viacep.com.br/ws/87114655/json/'); *//*

  cep.then((*response*) => {

    response.headers.forEach(console.log)

    console.log(response.status) *// verifica o status do fetch*

    });

**10.2.f URL**

Retorna o URL do fetch

*const* cep = fetch('https://viacep.com.br/ws/87114655/json/'); *//*

  cep.then((*response*) => {

    response.headers.forEach(console.log)

    console.log(response.url) *// verifica o status do fetch*

    });

**10.2.g Type**

*const* cep = fetch('https://viacep.com.br/ws/87114655/json/'); *//*

  cep.then((*response*) => {

    response.headers.forEach(console.log)

    console.log(response.type) *// verifica o tipo;*

    });

*// basic = proprio servidor // cors = requisição de outro servidor, mas permitida*

*// caso de erro, é porque o site não permite, pois não é uma api*

**10.2.h Exercicios fetch**

*// Utilizando a API https://viacep.com.br/ws/${CEP}/json/*

*// crie um formulário onde o usuário pode digitar o cep*

*// e o endereço completo é retornado ao clicar em buscar*

*const* botao = document.querySelector(".pesquisar");

botao.addEventListener("click", handleclick);

*function* handleclick(*event*) {

  event.preventDefault();

*const* cepEdit = document.querySelector("#cep");

*const* cep = cepEdit.value;

*const* cep2 = fetch(**`**https://viacep.com.br/ws/${cep}/json/**`**);

  cep2

    .then((*response*) => {

      return response.text();

    })

    .then((*response*) => {

*const* div = document.querySelector(".end");

      div.innerText = response;

    });

}

*// Utilizando a API https://blockchain.info/ticker*

*// retorne no DOM o valor de compra da bitcoin and reais.*

*// atualize este valor a cada 30s*

*function* btc() {

*const* bit = fetch("https://blockchain.info/ticker");

  bit

    .then((*response*) => {

      return response.json();

    })

    .then((*response*) => {

      return response.BRL.buy;

    })

    .then((*response*) => {

*const* money = response.toLocaleString("pt-BR", {

        style: "currency",

        currency: "BRL",

      });

*const* div2 = document.querySelector(".bit");

      div2.innerText = "Valor de compra da BitCoin " + money;

    });

}

btc()

setInterval(*function* () {

  btc();

}, 10000);

*// Utilizando a API https://api.chucknorris.io/jokes/random*

*// retorne uma piada randomica do chucknorris, toda vez que*

*// clicar em próxima*

  divPiada = document.querySelector('.piada')

*function* piada() {

*const* piada = fetch('https://api.chucknorris.io/jokes/random')

.then((*response*) => {

  return response.json();

}).then((*response*) => {

  divPiada.innerText = response.value

})

}

piada()

*const* btn = document.querySelector('.proxima')

btn.addEventListener('click', piada)

**10.3 Jason 2**

JavaScript Object Notation (JSON) é um formato de organização de dados, compostos por um conjunto de chave e valor. As aspas duplas são obrigatórias, tanto na chave quanto no valor quando este for uma string.

Os valores podem ser números, strings, boolean, arrays, objetos e null.

{

  "id": 1, *// pode ser um id*

  "faculdade": true, *// booleano*

  "pertences": [ *//array*

    "lapis",

    "caneta",

    "caderno" *// no ultimo , obrigatorio não ter virgula*

  ],

  "endereco": { *// pode ser um objeto*

    "cidade": "Rio de Janeiro",

    "pais": "Brasil"

  },

  "casado": null

}

É comum possuirmos array's com objetos em cada valor da array. Cuidado para não colocar vírgula no último item do objeto ou array.

**10.3.a Criando Array com objetos**

[

  {

    "id": 1,

    "aula": "JavaScript",

    "tempo": "25min"

  },

  {

    "id": 2,

    "aula": "HTML",

    "tempo": "15min"

  }

]

fetch('./dados.json')

.then (*r* => r.json()) *// Transofmra em objeto*

.then(*json* => {

  json.forEach(*materia* => {

    console.log(materia.tempo) *// puxando todos tempos*

  });

})

fetch('./dados.json')

.then (*r* => r.json()) *// Transofmra em objeto*

.then(*json* => {

  console.log(json[0].aula) *// Acessando o item 1 e a propriedade aula*

})

**10.3.b Metodo Parse para transformar text em objeto**

fetch('./dados.json')

.then (*r* => r.text())

.then(*jsonText* => {

*const* jsonFinal = JSON.parse(jsonText) *// outro metodo de transformar em objeto*

  console.log(jsonFinal)

})

**10.3.c Transformando objeto em arquivo json**

*const* config = {

  player: "Google",

  tempo: "25.5",

  aula: "2.1 JS"

}

*const* stringConfig = JSON.stringify(config)

console.log(stringConfig) *// {"player":"Google","tempo":"25.5","aula":"2.1 JS"}*

**10.3.d Enviando objeto para o local storage como json**

*const* config = {

  player: "Google",

  tempo: "25.5",

  aula: "2.1 JS"

}

localStorage.config = JSON.stringify(config)

**10.3.e voltando como objeto utilizando o parse**

console.log(JSON.parse(localStorage.config)) *// mostra como objeto*

**10.4 API**

* Application

Um servidor, aplicativo, objeto JavaScript ou qualquer outra coisa que você interaja através de comandos. Ao digitar uma URL, estamos utilizando a API do browser para se comunicar com a API do servidor.

* Programming

Programação, isso significa que um comando irá encadear uma cadeia de eventos pré-definidos. O resultado esperado é geralmente o mesmo.

* Interface

A interface são os comandos criados para permitir a interação com a aplicação. Ex: 'VIOLAO'.toLowerCase() é um método que faz parte da interface do objeto String. A interação com a interface retorna um efeito / resposta.

Api é a cadeia de eventos ao realizar um comando, por exemplo ao acessar um link, o servidor te envia um json com os dados solicitados

## EXEMPLOS DE API'S

* GitHub

https://api.github.com/users/origamid

https://api.github.com/users/origamid/followers

* Array / Element

[].map();

[].filter();

Element.classList;

Element.attributes;

* Tempo

https://www.metaweather.com/api/location/455825/

https://github.com/toddmotto/public-apis

**10.5 HTTP**

Hypertext Transfer Protocol é o protocolo utilizando para enviarmos/recebermos arquivos e dados na Web

**10.6 URL E METHOD**

Uma requisição HTTP é feita através de uma URL. O método padrão é o GET, mas existem outros como POST, UPDATE, DELETE, HEADER.

**10.6.a GET**

GET irá puxar as informações da URL. Não é necessário informar que o método é GET, pois este é o padrão.

*const* url = 'https://jsonplaceholder.typicode.com/posts/';

fetch(url, {

  method: 'GET'

})

.then(*r* => r.json())

.then(*r* => console.log(r))

**10.6.b POST**

POST irá criar uma nova postagem, utilizando o tipo de conteúdo especificado no headers e utilizando o conteúdo do body.

*const* url = 'https://jsonplaceholder.typicode.com/posts/'

*const* options = {

  method: 'POST', *//  envia para o servidor*

  body: '{"title": "javaScript"}', *// conteudo a ser enviado*

  headers: {

    "Content-Type": "application/json; charset=utf-8", *// tipo de conteudo*

  }

}

**10.6.cPUT**

PUT irá atualizar o conteúdo do URL com o que for informado no conteúdo do body.

*const* url = 'https://jsonplaceholder.typicode.com/posts/1/'; *// informando qual conteudo será atualizado no link*

fetch(url, {

  method: 'PUT',

  headers: {

    "Content-Type": "application/json; charset=utf-8",

  },

  body: '{"titulo": "JavaScript"}'

})

**10.6.d HEAD**

HEAD puxa apenas os headers. É uma requisição mais leve pois não puxa o body.

*const* url = 'https://jsonplaceholder.typicode.com/posts/1/';

fetch(url, {

  method: 'HEAD',

})

.then(*response* => {

  response.headers.forEach(console.log);

  console.log(response.headers.get('Content-Type')); *// exibindo o head*

});

**HEADERS**

* Cache-Control

Tempo que o arquivo deve ficar em cache em segundos. Ex: public, max-age=3600

* Content-Type

Tipo de conteúdo. Ex: text/html; charset=utf-8. Indicar o tipo de arquivo principalmente em métodos POST e PUT.

* Lista de Headers

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

**10.6.e Cors**

Cross-Origin Resource Sharing, gerencia como deve ser o compartilhamento de recursos entre diferente origens.

É definido no servidor se é permitido ou não o acesso dos recursos através de scripts por outros sites. Utilizando o Access-Control-Allow-Origin.

Se o servidor não permitir o acesso, este será bloqueado. É possível passar por cima do bloqueio utilizando um proxy.

CORS é um acordo entre browser / servidor ou servidor / servidor. Ele serve para dar certa proteção ao browser, mas não é inviolável.

**10.7 Async Await**

A palavra chave async indica que a função possui partes assíncronas e que você pretende esperar a resolução da mesma antes de continuar. O await irá indicar a promise que devemos esperar. Faz parte do ES8.

*// antes da função deve-se declarar que é async e o await antes do fetch*

async *function* puxarDados() {

*const*  responseDados = await fetch('./dados.json') *// espera a promessa*

*const* jsonDados = await responseDados.json() *// espera o resultado da promessa*

  console.log(jsonDados) *// retorna o objeto do json*

  document.body.innerText = jsonDados.aula *// utiliza a propriedade 'aula' do objeto e envia para o html*

}

puxarDados()

**10.7.a Then e Async**

A diferença é uma sintaxe mais limpa

*function* iniciarFetch() {

  fetch('./dados.json')

  .then(*dadosResponse* => dadosResponse.json())

  .then(*dadosJSON* => {

    document.body.innerText = dadosJSON.titulo;

  })

}

iniciarFetch();

async *function* iniciarAsync() {

*const* dadosResponse = await fetch('./dados.json');

*const* dadosJSON = await dadosResponse.json();

  document.body.innerText = dadosJSON.titulo;

}

iniciarAsync();

**10.7.b Try e Catch em async**

Para lidarmos com erros nas promises, podemos utilizar o try e o catch na função.

async *function* puxarDados() {

  try { *// tente*

*const* responseDados = await fetch("./dados.json"); *// espera a promessa*

*const* jsonDados = await responseDados.json(); *// espera o resultado da promessa*

    console.log(jsonDados); *// retorna o objeto do json*

    document.body.innerText = jsonDados.aula; *// utiliza a propriedade 'aula' do objeto e envia para o html*

  } catch { *// caso erro, execute;*

    console.log("Erro");

  }

}

puxarDados();

**10.7.c iniciando fetch ao mesmo tempo**

Não precisamos esperar um fetch para começarmos outro. Porém precisamos esperar a resposta resolvida do fetch para transformarmos a response em json

async *function* puxarDados() {

*const* promiseDados = fetch("./dados.json");  *// promessa de dados*

*const* promiseclientes = fetch("./clientes.json");

*const* jsonDados = await (await promiseDados).json(); *// espera para promessa dados ser recebido*

*const* jsonclientes = await (await promiseclientes).json();

    console.log(jsonDados)

    console.log(jsonclientes)

  }

puxarDados();

**10.7.d o resultando a frente do await deve ser uma promisse, caso não seja o await não tem efeito**

async *function* asyncSemPromise() {

*// Console não irá esperar.*

  await setTimeout(() => console.log('Depois de 1s'), 1000);

  console.log('acabou');

}

asyncSemPromise();

async *function* iniciarAsync() {

  await new Promise(*resolve* => {  *// criando promessa*

    setTimeout(() => resolve(), 1000) *// após um seg*

  });

  console.log('Depois de 1s'); *// é executada a função*

*const* lida = 5 + 5

  console.log(lida)

}

iniciarAsync();

**10.8 History API**

É possível acessarmos o histórico de acesso do browser em uma sessão específica através do window.history. O objeto history possui diferentes métodos e propriedades.

window.history;

window.history.back(); *// vai para a anterior*

window.history.forward(); *// vai para a próxima*

**10.8.a PushState**

A parte interessante de manipularmos o history é que podemos modificar o histórico e adicionar um novo item. window.history.pushState(obj, title, url).

*// Em obj podemos enviar um objeto com dados*

*// mas o seu uso é restrito por isso geralmente utilizamos*

*// null. O title ainda é ignorado por alguns browsers, também*

*// utilizamos null nele. O url que é parte importante.*

window.history.pushState(null, null, 'sobre.html');

**10.8.b PopState**

O evento popstate pode ser adicionado ao objeto window. Assim podemos executar uma função toda vez que o usuário clicar no botão de voltar ou próximo.

**10.8.c FETCH E HISTORY**

Ao puxarmos dados via fetch api, o url da página continua o mesmo. Ao combinar fetch com a history api conseguimos simular uma navegação real entre páginas, sem a necessidade de recarregamento da mesma.

async *function* fetchPage(*url*) {

*const* pageReponse = await fetch(url);

*const* pageText = await pageReponse.text();

  window.history.pushState(null, null, url);

}

**10.9 – Trocando Html da pag atual**

trocar uma section da pagina, sem que seja necessário a troca de toda a pagina

Podemos utilizar o fatch para receber um html ao clicar em um link, e então substituir partes do html atual

*const* links = document.querySelectorAll('a') *//1- seleciona todos os links da pagina atual*

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

  link.addEventListener('click', handleclick) *//2 para cada link, temos uma function handleclick*

})

*function* handleclick(*event*) { *//3 função que irá receber o html*

  event.preventDefault() *//4 previnindo o padrão do link*

  fetchPage(event.target.href) *//5 aciona a função de troca enviando o link da pag selecionada*

  window.history.pushState(null,null, event.target.href) *//6 trocalos o link da pag atual pelo da pagina clicada*

}

async *function* fetchPage(*url*) { *//7 function que recebe o html, async pois tem await para receber o html*

  document.querySelector('.content').innerHTML ='Carregando'*//8 Enquanto a pagina não carrega, pode ser incluso algum aviso de carregando*

*const* pageResponse = await fetch(url) *//9 recebendo com fetch todo o html, a function não seguira enquanto não receber, por conta do await*

*const* pageText = await pageResponse.text()   *//10 recebido, transforma em text*

  replaceContent(pageText) *//11 aciona a função de substituição de conteudo, enviando em formato de texto*

}

*function* replaceContent(*newText*) {*//12 funcão de substituição, recebenndo o html da pag selecionada*

*const* newHtml = document.createElement('div') *//13 cria uma div no document*

  newHtml.innerHTML = newText *//14 a div recebe todo o html*

*const* oldContent = document.querySelector('.content') *//15 seleciona o content da pag atual*

*const* newContent = newHtml.querySelector('.content') *//16 seleciona o content da pag clicada*

  oldContent.innerHTML = newContent.innerHTML *//17 substitui o conteudo 'content' da pag atual pelo da clicada*

  document.title = newHtml.querySelector('title').innerText *//18 o titulo atual recebe o innerText da tag title da pag clicada*

*//obs: pode realizar quantas trocas necessario*

}

window.addEventListener('popstate', () => {*//19 adionando eventlistener na window, para quando clicar em avancar ou voltar*

  fetchPage(window.location.href) *//20 quando o usuario clicar em voltar, executamos a função fetch enviando o link atual como parametro*

} )

**11.1 inserindo informações do json no html**

utilizando fetch para importar numeros de uma api que poderia ser de algum servidor,

vamos criar o HTML e inserir os dados da API

import initAnimaNumeros from "./initAnimaNumeros.js"; *//puxando a animação dos numeros para ser iniciada após o fetch*

export default *function* initAnimaisFetch() {

async *function* fetchAnimais(*url*) { *//2 function que irá fazer o fetch, ativar a função de troca, e inserir no html*

  try {

*const* animaisResponse = await fetch(url) *//3 realiza o fetch da api passada na chamada a function*

*const* animaisJson = await animaisResponse.json() *//4 transforma em json*

*const* numerosGrid = document.querySelector('.numeros-grid') *//5 seleciona a div que irá receber os dados*

  animaisJson.forEach(*animal* => { *//6 para cada objeto dentro do json*

*const* divAnimal = createAnimal(animal) *//7 ativamos a função que irá criar o html*

      numerosGrid.appendChild(divAnimal) *//13 adiciona o html criado dentro da div selecionada*

  });

  initAnimaNumeros() *//14 inicia a function importada de animação dos numeros*

} catch(erro) {

  console.log(erro)

}

}

*function* createAnimal(*animal*) { *//8 função que cria o html com as informações do json*

*const* div = document.createElement('div'); *//9 cria um elemento div*

div.classList.add('numero-animal'); *//10 adiciona a classe*

div.innerHTML = **`**<h3>${animal.especie}</h3><span data-numero="">${animal.total}</span>**`** *//11 conteudo html da div, com um h3 e um span*

 return div *//12 retorna o conteudo html para ser inserido no html*

}

fetchAnimais('./numeros.json') *//1 chamando a função dom o link da "API"*

}

**11.2.a Api puxando valor de bitcoin**

export default *function* initBitcoinFetch() {

*//fetch de api utilizando then*

  fetch("https://blockchain.info/ticker")

  .then((*response*) => response.json()) *//transforma a api em json*

  .then((*json*) => {

*const* btc = document.querySelector(".btcPreco"); *// seleciona a area que ira ser inserido*

    btc.innerText = (1000 / json.BRL.sell).toFixed(4); *// divide uma api por 1000 e fixa em 4 casas*

  })

  .catch((*erro*) => {

    console.log(Error(erro)); *// caso de erro apresenta erro no log*

  });

}

**11.0 Classes**

**11.0.a Constructor function**

*// Criando objeto com a função construtora*

*const* blueButton = new Button('Cpmprar', 'blue')

*// Objeto criado com a função construtora utilizando o metodo de criação de button*

console.log(blueButton.element())

*//Função construtora comum*

*function* Button(*text*, *background*) {

  this.text = text

  this.background = background

  }

*// Criando metodo para a função construtora*

  Button.prototype.element = *function*() {

*const* buttonElement = document.createElement('button')

    buttonElement.innerText = this.text

    buttonElement.style.background = this.background

    return buttonElement

  }

*// Criando objeto com a função construtora*

*const* blueButton = new Button('Cpmprar', 'blue')

*// Objeto criado com a função construtora utilizando o metodo de criação de button*

  console.log(blueButton.element())

**11.0.b Class**

O ES6 trouxe uma nova sintaxe para implementarmos funções construtoras. Agora podemos utilizar a palavra chave class. É considerada syntactical sugar, pois por baixo dos panos continua utilizado o sistema de protótipos de uma função construtora para criar a classe.

*class* Button { *// contrutora*

*constructor*(*text*, *background*) { *// parametros ficam dentro de constructor*

    this.text = text;

    this.background = background;

  }

  element() { *// metodos ficam dentro da class*

*const* buttonElement = document.createElement('button');

    buttonElement.innerText = this.text;

    buttonElement.style.background = this.background;

    return buttonElement;

  }

}

*const* blueButton = new Button('Comprar', 'blue');

**11.0.c Constructor**

O método constructor(args) {} é um método especial de uma classe. Nele você irá definir todas as propriedades do objeto que será criado. Os argumentos passados em new, vão direto para o constructor.

*class* Button {

*constructor*(*text*, *background*, *color*) {

    this.text = text;

    this.background = background;

    this.color = color;

  }

}

*const* blueButton = new Button('Clique', 'blue', 'white');

*// Button {text: 'Clique', background: 'blue', color: 'white'}*

**11.0.d Exemplo Função construtora class**

*//Função construtora class*

*class* Button {

*constructor* (*text*, *background*) { *//parametros da construção*

this.text = text

this.background = background

}

element() { *// metodo para criar o elemento*

*const* buttonElement = document.createElement('button') *// cria o botao*

  buttonElement.innerText = this.text *// adiciona texto ao botao*

  buttonElement.style.background = this.background *// adiciona background*

  return buttonElement *// retorna o elemento*

}

appendTo(*target*) { *// metodo que utiliza o metodo de criação de elemento, e adiciona no html*

*const* targetElement = document.querySelector(target) *// selecionando o elemento com o parametro body*

  targetElement.appendChild(this.element()) *// utilizando this. e acionando o metodo element*

  return targetElement *// adiciona no body*

}

}

*const* blueButton = new Button('Comprar', 'blue')

console.log(blueButton.element(this.element)) *// retorna o elemento*

console.log(blueButton) *// retorna o objeto*

console.log(blueButton.appendTo('body')) *// utiliza a const blueButton e o metodo que adiciona ao html*

**11.0.e objeto em uma propriedade**

*class* Button {

*constructor*(*options*) {

    this.options = options;

  }

}

*const* blueOptions = {

  backgroundColor: 'blue',

  color: 'white',

  text: 'Clique',

  borderRadius: '4px',

}

*const* blueButton = new Button(blueOptions);

console.log(blueButton.options) *// objeto*

console.log(blueButton.options.backgroundColor) *// item do objeto*

**11.0.f Class com static**

Por padrão todos os métodos criados dentro da classe irão para o protótipo da mesma. Porém podemos criar métodos diretamente na classe utilizando a palavra chave static.

*class* Button {

*constructor*(*options*) {

    this.options = options;

  }

  static createButton(*text*, *background*) { *// criando metodo na class*

*const* buttonElement = document.createElement('button')

    buttonElement.innerText = text

    buttonElement.style.background = background

    return buttonElement

  }

}

*const* blueOptions = {

  backgroundColor: 'blue',

  color: 'white',

  text: 'Clique',

  borderRadius: '4px',

}

*const* blueButton = new Button(blueOptions);

*const* blueStatic = Button.createButton('clique', 'blue') *//utilizando createButton como metodo de button(class)*

**11.0.g Static – Retornando pré definições da class**

*class* Button {

*constructor*(*text*, *background*) {

    this.text = text;

    this.background = background;

  }

  element() {

*const* elementButton = document.createElement('button');

    elementButton.innerText = this.text;

    elementButton.style.background = this.background;

    return elementButton

  }

  static createBlue(*text*) {

    return new Button(text, 'blue'); *// pré define a propriedade background para blue*

  }

}

*const* blueButton = Button.createBlue('Comprar'); *// passaando apenas o texto*

console.log(blueButton) *// retorna o objeto com comprar e blue*

**11.1 Get e Set em class**

Podemos definir comportamentos diferentes de get e set para um método.

**11.1.a Get – valor estatico**

Se definirmos apenas o get de um método, teremos então um valor estático que não será possível mudarmos.

*const* matematica = {

*get* PI() {

    return 3.14;

  },

};

matematica.PI; *// get (3.14)*

matematica.PI = 20; *// nada acontece*

**11.1.b Set – setar valores**

Eu posso ter um método com set apenas, que modifique outras propriedades do meu objeto.

*const* frutas = {

  lista: [],

*set* nova(*fruta*) {

    this.lista.push(fruta);

  },

};

frutas.nova = 'Banana';

frutas.nova = 'Morango';

frutas.lista; *// ['Banana', Morango];*

**11.1.c Class e get set**

Assim como em um objeto, as classes podem ter métodos de get e set também.  
Com o set podemos modificar apenas parte do elemento de get. É comum definirmos variáveis privadas, utilizando o underscore \_privada.

*//1 class construtora*

*class* Button1 {

*constructor*(*text*, *color*) {

    this.text = text,

    this.color = color

  }

*get* element() {*//2 get- irá receber os parametros e setar ou não o type*

*const* type = this.\_elementType || 'button' *//3  caso não for definido valor, considerado button*

*const* buttonElement = document.createElement(type) *//4 poderá ou nao receber valor*

    buttonElement.innerText = this.text

    buttonElement.style.color = this.color

    return buttonElement

  }

*set* element(*type*) {

    this.\_elementType = type *//5 atribui valor, caso informado*

  }

}

*const* blueButton = new Button1('comprar', 'blue')

console.log(blueButton.element) *//6 retorna button por padrão*

blueButton.element = 'div' *//7 setando com set o valor div*

console.log(blueButton.element) *//8 retorna div*

**11.1.d Class extends**

Class sendo um constructor, Podemos criar o class padrão, com subclass para derivados,   
o objeto criado terá tanto os métodos da class quanto da subclasse

*//class construtora*

*class* Veiculo {

*constructor*(*rodas*) {

    this.rodas = rodas

  }

  acelerar() {

    console.log('Acelerou')

  }

}

*// class construtora extends*

*class* Moto extends Veiculo {

  acelerar() { *// quando um metodo se repete no extended, objetos criados com esse extended irão utilizar esse metodo, e não o da class*

    console.log('acelerou rapido')

  }

  empinar() {

    console.log('Empinou com ' + this.rodas + ' rodas') *// o extends tem acesso ao this*

  }

}

*// criando a class com o extends ela tem o metodo referente*

*const* honda = new Moto(2) *// const honda tem os metodos acelerar e empinar*

console.log(honda.empinar()) *//Empinou com 2 rodas*

console.log(honda.acelerar()) *// acelerou rapido*

*// caso criamos com a class ela não terá o metodo extends;*

*const* civic = new Veiculo(4)

console.log(civic.acelerar())

*//console.log(civic.empinar()) // não tem o metodo*

**11.1.e Class extends Super**

Com o super dentro do método, falamos com a class, e podemos executar o método da class + o método do extends

*class* Moto extends Veiculo {

  acelerar() {

*super*.acelerar()

    console.log('acelerou rapido')

  }

  empinar() {

    console.log('Empinou com ' + this.rodas + ' rodas')

  }

}

*const* honda = new Moto(2)

console.log(honda.acelerar()) *// executa os dois metodos*

**11.1.f Class extends Super propriedades**

Podemos utilizar o super para estendermos o método constructor.

*// Neste exemplo extendemos o constructor de moto para ter a propriedade capacete,*

*class* Moto extends Veiculo {

*constructor*(*rodas*, *capacete*) { *// necessario passar os parametros da class*

*super*(rodas) *// parametros da class passado no super*

    this.capacete = capacete *// novo parametro*

  }

  acelerar() {

*super*.acelerar()

    console.log('acelerou rapido')

  }

  empinar() {

    console.log('Empinou com ' + this.rodas + ' rodas')

  }

}

*const* honda = new Moto(2, true) *// parametro da class e extends*

console.log(honda) *//Moto {rodas: 2, capacete: true} // parametro capacete adicionado*

class Veiculo {

constructor(rodas, combustivel) {

this.rodas = rodas;

this.combustivel = combustivel;

}

}

class Moto extends Veiculo {

constructor(rodas, combustivel, capacete) {

super(rodas, combustivel);

this.capacete = capacete;

}

}

const honda = new Moto(4, 'Gasolina', true);

**11.2 Plugin Count Down**

Aqui vamos criar uma class que irá receber uma data futura, e realizar uma contagem de quanto tempo falta até ela

A const que chamar a construtora irá receber todos os métodos, como propriedade

import CountDown from './countdown.js' *// importanto o plugin de contagem*

*// criando um objeto com a construtora class, passando como parametro a data desejada*

*const* TempoParaNatal = new CountDown('24 December 2023 23:59:59 GMT-0300')

*const* TempoParaAnoNovo = new CountDown('31 December 2023 23:59:59 GMT-0300')

*// metodo sendo utilizado como propriedade*

*//pois agora é uma propriedade construida com o class*

console.log(TempoParaNatal.total)

console.log(TempoParaAnoNovo.total)

console.log(TempoParaNatal)

export default *class* CountDown { *// exportando o plugin*

*constructor*(*futureDate*) { *// recebendo como parametro a data passada*

    this.futureDate = futureDate;

  }

*get* \_actualDate() {

    return new Date();*// colocando a data atual em um metodo interno*

  }

*get* \_futureDate() {

    return new Date(this.futureDate); *// data do parametro em um metodo interno*

  }

*get* \_timeStampDiff() {

    return this.\_futureDate.getTime() - this.\_actualDate.getTime(); *// diferença até a data do parametro*

  }

*get* days() {

    return Math.floor(this.\_timeStampDiff / (24 \* 60 \* 60 \* 1000)); *// utilizando math para arredondar*

  }

*get* horas() {

    return Math.floor(this.\_timeStampDiff / (60 \* 60 \* 1000));

  }

*get* minutos() {

    return Math.floor(this.\_timeStampDiff / (60 \* 1000));

  }

*get* segundos() {

    return Math.floor(this.\_timeStampDiff / 1000);

  }

*get* total() {

*const* days = this.days;

*const* hours = this.horas % 24; *// restante da divisão por 24*

*const* minutes = this.minutos % 60

*const* seconds = this.segundos % 60

    return { *// metodo total retornando um objeto*

      days,

      hours,

      minutes,

      seconds,

    }

  }

}

**12.0 Mais JavaScript**

**12.1 Function declaration**

*function* somar(*a*,*b*) { *//declarado com function e o nome da função*

  return a + b;

}

somar(4,2); *// 6*

12.2 Function Expression

É criada a partir da declaração de uma variável, na qual assinalamos uma função. Esta função pode ser anônima ou nomeada. A mesma poderá ser ativada através da variável assinalada.

*const* somar = *function*(*a*,*b*) {

  return a + b;

}

somar(4,2); *// 6*

**12.2.a Hoisting**

Function Declarations são completamente definidas no momento do hoisting, já function expressions apenas serão definidas no momento da execução. Por isso a ordem da declaração de uma FE possui importância.

somar(4,2); *// 6 // pois function declaration é executada*

dividir(4,2); *// Erro // function expression necessida ter a chamada após a function*

*function* somar(*a*,*b*) {

  return a + b;

}

*const* dividir = *function*(*a*,*b*) {

  return a / b;

}

**12.2.b Arrow Function - expression**

*const* somar = (*a*,*b*) => {

  return a + b;

}

console.log(somar(4,5))

*//Caso for retornar apenas uma linha, é possivel reduzir o codigo, porem não é recomendado*

*// const somar = (a,b) => a + b;*

**12.2.c Estrutura / Preferencia**

Geralmente o uso entre expression / declaration é uma questão de preferência. Function Expression força a criação da mesma antes de sua ativação, o que pode contribuir para um código mais estruturado.

Ao utilizar a function declaration em um script que não é um modulo, ela entrara como método no objeto window, podendo haver conflitos com outros métodos, ou entrar em conflito com plugins trazidos para dentro do código

Por isso é indicado utilizar o function expression.

12.2.d IIFE - IMMEDIATELY INVOKED FUNCTION EXPRESSION

Antes da introdução de modules e da implementação do escopo de bloco, a forma mais comum utilizada para isolarmos o escopo de um código JavaScript era através das chamadas IIFE's.

*var* instrumento = 'Violão';

(*function*() {

*// código isolado do escopo global*

*var* instrumento = 'Guitarra';

  console.log(instrumento); *// Guitarra*

})();

console.log(instrumento); *// Violão*

**12.3 Factory Function**

São funções que retornam um objeto sem a necessidade de utilizarmos a palavra chave new. Possuem basicamente a mesma função que constructor functions / classes.

Não é necessário declarar com new a factory function

*function* createButton(*text*) {

*function* createElement() {

*const* buttonElement = document.createElement('button')

      buttonElement.innerText = text

      return buttonElement

    }

    return {

      text,

      createElement

    }

}

*const* btnComprar = createButton('comprar')

console.log(btnComprar) *// retorna um objeto com o parametro e uma função*

console.log(btnComprar.createElement()) *// cria o botão*

*const* btnVender = createButton('Vender')

console.log(btnVender)

console.log(btnVender.createElement())

**12.3.a Metodos e variáveis privadas**

Uma das vantagens é a possibilidade de criarmos métodos / variáveis privadas.

Só é utilizável o que é apresentado no return

**12.3.b Congelando o retorno**

Ao utilizar o object.freeze o usuário é incapaz de mudar o parâmetro de retorno

    return Object.freeze({

      text,

      createElement

    })

**12.3.c Constructor function Sem o new**

Uma das vantagens da Factory Function é a possibilidade de iniciarmos a mesma sem a utilização da palavra chave new. Também é possível fazer isso com uma Constructor Function.

*// caso a declaração não tiver o new*

*//realizamos uma verificação para saber se pessoa(com new) ou window(sem pessoa)*

*//caso a declaração não tiver o new, retorna com new*

*function* Pessoa(*nome*) {

  if (!(this instanceof Pessoa)) *// novo metodo: (!new.target)*

    return new Pessoa(nome);

  this.nome = nome;

}

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

  return **`**${this.nome} andou**`**;

}

*const* designer = Pessoa('André');

**12.3.d Uso real factory function**

*// função será utilizada para ter diversos metodos, para o elemento passado como parametro*

*function* $$(*selectedElements*) {  *// $$ nada mais é que um nome de função*

*const* elements = document.querySelectorAll(selectedElements); *// seleciona o elemento conforme parametro passado*

*function* hide() {

    elements.forEach((*element*) => {

      element.style.display = "none";

    });

    return this; *// após cada metodo, devemos retornar o this, para que possa ser utilizado o proximo metodo*

  }

*function* show() {

    elements.forEach((*element*) => {

      element.style.display = "initial"; *// retorna ao display inicial, antes do none*

    });

    return this;

  }

*function* on(*onEvent*, *callback*) {

    elements.forEach((*element*) => {

      element.addEventListener(onEvent, callback); *// podemos passar como parametro o tipo de evento, e o metodo/função que realizara no callback*

    });

    return this;

  }

*function* addClass(*className*) {

    elements.forEach((*element*) => {

      element.classList.add(className);

    });

    return this;

  }

*function* removeClass(*className*) {

    elements.forEach((*element*) => {

      element.classList.remove(className);

    });

    return this;

  }

  return { *// todos metodos listados no return*

    elements,

    hide,

    show,

    on,

    addClass,

    removeClass,

  };

}

*// é possivel chamar diversos parametros*

buttons.hide().show().addClass('ativo').removeClass('ativo');

*const* btns = $$("button"); *// chamando função com parametro tag button*

*function* handleClick(*event*) {

  console.log(event.target);

}

btns.on("click", handleClick);

btns.addClass("ativo");

btns.removeClass("ativo");

**12.4 Escopo e closures**

A funcao2 possui 4 escopos. O primeiro escopo é o Local, com acesso ao item3. O segundo escopo dá acesso ao item2, esse escopo é chamado de Clojure (funcao1) (escopo de função dentro de função). O terceiro escopo é o Script com acesso ao item1 e o quarto escopo é o Global/Window.

*let* item1 = 1;

*function* funcao1() {

*let* item2 = 2;

*function* funcao2() {

*let* item3 = 3;

    console.log(item1);

    console.log(item2);

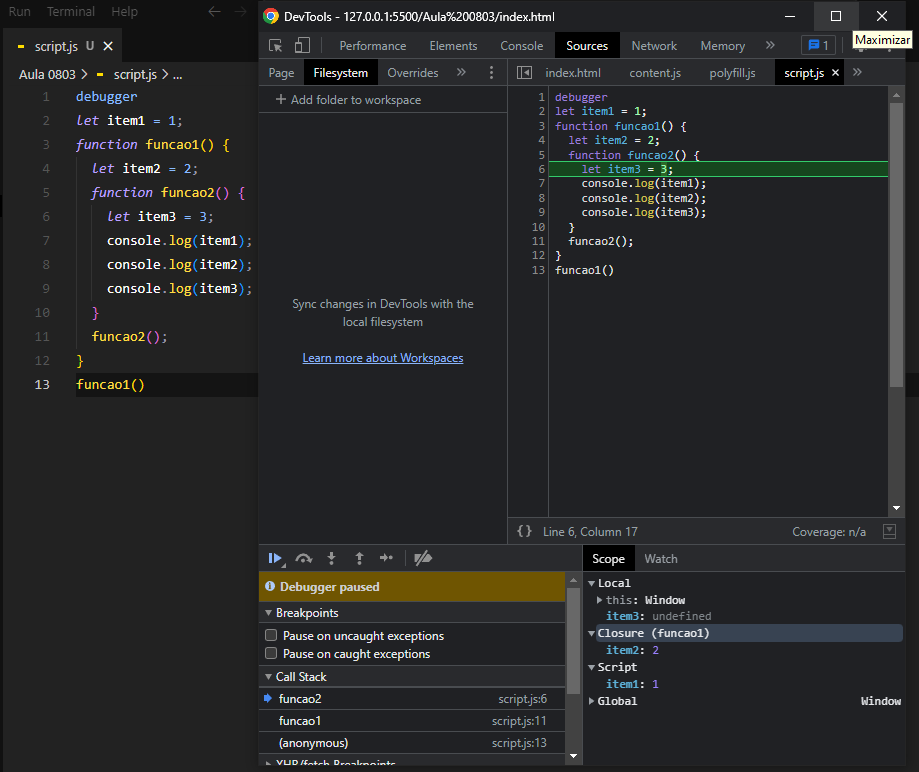
    console.log(item3);

  }

  funcao2();

}

**12.5 Debugger**



**12.6 Destructuring**Cria uma nova const com o dado de um objeto

**12.6.a Destructuring objects**A desestruturação irá facilitar a manipulação de dados. Principalmente quando temos uma grande profundidade de objetos.

O nome das variáveis DEVE ser o mesmo que o de dentro dos objetos,

Mas podemos modificar ela;

*const* {fisicas: CompraPresencial,

console.log(CompraPresencial);

Valor prédefinido

Caso uma propriedade não tenha sido setado valor, podemos definir um valor predefinido

      email: [],

email = 'email@gmail.com'

*const* cliente = {

  nome: 'Andre',

  compras: {

    digitais: {

      livros: ['Livro 1', 'Livro 2'],

      videos: ['Video JS', 'Video HTML']

    },

    fisicas: {

      cadernos: ['Caderno 1']

    }

  }

}

*const* {livros, videos} = cliente.compras.digitais; *// const livros e videos recebe as duas propriedades do objeto digitais*

console.log(livros);

console.log(videos);

*// modo sem a desestruturização*

*//console.log(cliente.compras.digitais.livros);*

*//console.log(cliente.compras.digitais.videos);*

**12.6.b Nesting**É possível aninhar uma desestruturação dentro de outra.

*const* {fisicas, digitais, digitais: {livros, videos}} = cliente.compras;  *// acessando o nivel digitais*

console.log(fisicas);

console.log(digitais);

console.log(livros);

console.log(videos);

**12.6.c DESTRUCTURING ARRAYS**

Para desestruturar array's você deve colocar as variáveis entre [] colchetes.

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

*/\* sem destructuring*

*const primeiraFruta = frutas[0];*

*const segundaFruta = frutas[1];*

*const terceiraFruta = frutas[2];*

*\*/*

*// Com destructuring*

*const* [primeira, segunda, terceira] = frutas; *//como é por numero, não é necessario informar o nome*

**12.6.d Declarações múltiplas de variável**

*const* primeiro = 'Item 1';

*const* segundo = 'Item 2';

*const* terceiro = 'Item 3';

*// ou*

*const* [primeiro, segundo, terceiro] = ['Item 1', 'Item 2', 'Item 3'];

*// Troque os valores das variáveis abaixo*

*let* cursoAtivo = 'JavaScript';

*let* cursoInativo = 'HTML';

[cursoAtivo, cursoInativo] = [cursoInativo,cursoAtivo]

**12.6.e Destructuring de objetos**

Podemos utilizar as propriedades de um objeto

*function* keyUp({*key*, *keyCode*}) { *// desestruturando e selecionando o key do event keyup*

  console.log(key, keyCode) *//utilizando a propriedade key*

}

document.addEventListener('keyup', keyUp)

*function* handleKeyboard(*event*) {

  console.log(event.key);

}

*// Com Destructuring*

*function* handleKeyboard({*key*}) {

  console.log(key);

}

document.addEventListener('keyup', handleKeyboard);

**12.6.e getComputedStyles**

Retorna Valores definidos pelo navegador

*// Extraia o backgroundColor, color e margin do btn*

*const* btn = document.querySelector('button');

*const* btnStyles = window.getComputedStyle(btn);

console.log(btnStyles)

*const* {backgroundColor, margin} = btnStyles

console.log(backgroundColor, margin)

**12.7 Parametros**

Nem todos os parâmetros que definimos são utilizados quando uma função é executada, devido a falta de argumentos. Por isso é importante nos prepararmos para caso estes argumentos não sejam declarados

**12.7.a Parametro inicial – padrão**

Podemos definir parâmetros padrão, caso nenhuma seja definido passando diretamente no parametro

*function* perimetroForma(*lado*, *totalLados* = 4) { *// caso não seja passado parametro, considera 4*

  return lado \* totalLados;

}

perimetroForma(10, 5); *// 50*

perimetroForma(10); *// 40*

*// Reescreva a função utilizando*

*// valores iniciais de parâmetros com ES6*

*function* createButton(*background* = 'blue', *color* = 'red') {

*const* buttonElement = document.createElement('button');

  buttonElement.style.background = background;

  buttonElement.style.color = color;

  return buttonElement;

}

**12.7.b Arguments**

É possível puxar para uma const os argumentos utilizados na function com o arguments, que transforma os argumentos em uma array

*function* perimetroForma(*lado*, *totalLados* = 4) {

  console.log(*arguments*)

  return lado \* totalLados;

}

perimetroForma(10);

perimetroForma(10, 4, 20);

**12.7.c parâmetro rest – lista de argumentos**

É possível declararmos uma parâmetro utilizando ... na frente do mesmo. Assim todos os argumentos que passarmos na ativação da função, ficarão dentro do parâmetro.

*function* perimetroForma(*lado*, *totalLados* = 4, ...*listaArgumentos*) {

  console.log(listaArgumentos) *// [20, 'oi', 'teste']*

  console.log(*arguments*) *// Arguments(5) [10, 4, 20, 'oi', 'teste'*

  return lado \* totalLados;

}

console.log(perimetroForma(10, 4, 20, 'oi', 'teste'))

*// retorna a lista de argumentos*

*//e tambem retorna a soma dos parametros passados*

*function* anunciarGanhadores(*premio*, ...*ganhadores*) {

  ganhadores.forEach(*ganhador* => {

    console.log(ganhador + ' ganhou um ' + premio)

  });

}

anunciarGanhadores('Carro', 'Pedro', 'Marta', 'Maria');

**12.7.d Operador Spread**

Assim como o rest, o operador Spread também utiliza os ... para ser ativado. O spread irá distribuir um item iterável, um por um.

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

*const* legumes = ['Cenoura', 'Batata'];

*const* comidas = [...frutas, 'Pizza', ...legumes]; *// cria uma array utilizando as duas outras arrays*

*// Utilize o método push para inserir as frutas ao final de comidas.*

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

*const* comidas = ['Pizza', 'Batata'];

comidas.push(...frutas)

console.log(comidas) *//  ['Pizza', 'Batata', 'Banana', 'Uva', 'Morango']*

**12.7.e spread argument**

O Spread pode ser muito útil para funções que recebem uma lista de argumentos ao invés de uma array.

*const* numeroMaximo = Math.max(4,5,20,10,30,2,33,5); *// 33 // passando todos os numeros no verificador*

*const* listaNumeros = [1,13,21,12,55,2,3,43];

*const* numeroMaximoSpread = Math.max(...listaNumeros); *// passando no verificador como array*

console.log(listaNumeros) *//(8) [1, 13, 21, 12, 55, 2, 3, 43]*

console.log(numeroMaximoSpread) *//55*

*// //// //// //// //// //// //// //// //// //// //// //// //// //// //// //// //// ////*

*function* anunciarGanhadores(*premio*, ...*ganhadores*) {

  ganhadores.forEach(*ganhador* => {

    console.log(ganhador + ' ganhou um ' + premio)

  });

}

*const* ganhadores =  ['Pedro', 'Marta', 'Maria']L

anunciarGanhadores('Carro', ...ganhadores);

**12.7.f transformando em array com o spread**

É possível transformar itens iteráveis em uma array real com o spread.

*const* btns = document.querySelectorAll('button'); *// cria uma Nodelist*

*const* btnsArray = [...btns]; *// cria sem o arrayFrom uma array*

*const* frase = 'Isso é JavaScript';

*const* fraseArray = [...frase];

**12.8 Iterable**

São objetos que possuem o método [Symbol.iterator], geralmente no protótipo, é dentro dele que a função que lida com a iteração será definida. Ex: Array, String, NodeList, boa parte das Array-Like e outros.

**12.8.a For of**É possível fazemos um loop por cada iteração do objeto iterável utilizando o for...of. Além deste loop podemos também utilizar o Spread Operator nos mesmos.

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

*const* frase = 'Isso é JavaScript';

*// podemosutilizar o for off nesses itens pois eles tem Symbol(Symbol.iterator)*

for(*const* fruta of frutas) { *// para cada item de frutas*

  console.log(fruta); *//é realizado um log*

}

for(*const* char of frase) { *// para daca letra da string*

  console.log(char); *// é realizado um log*

}

*const* buttons = document.querySelectorAll('button');

//utilizando for of para setar cor

for (*const* btn of buttons) {

  btn.style.color = 'blue'

}

console.log(...buttons) *//node list, podemos utilizar pois tem  Symbol(Symbol.iterator)*

// nesse caso, não conseguimos utilizer o for of pois se trata de um objeto, que não tem o Symbol(Symbol.iterator)

*const* carro = {

  marca: 'Honda',

  ano: 2018,

}

*// Erro, não é Iterável*

for(*const* propriedade of carro) {

  console.log(propriedade); *// ript.js:28 Uncaught TypeError: carro is not iterable*

 }

**12.8.b For in**

Para objetos podemos utilizar o for in

*// metodo for in puxa apenas propriedades enumeradas*

*const* carro = {

  marca: 'Honda',

  ano: 2018,

}

Object.defineProperties(carro, {

  rodas: {

    value: 4,

    enumerable: true, *// é exibido no for in*

  },

  tetoSolar: {

    value: true,

    enumerable: false, *// não é exibido no for in*

  }

})

for(*const* propriedade in carro) {

  console.log(propriedade, carro[propriedade]);*//marca Honda  ano 2018 rodas 4*

 }

**12.8.c for in chave e valor**

Utilizando o for...in podemos retornar todas as chaves e valores de propriedades enumeráveis.

*const* btn = document.querySelector('button');

*const* btnStyles = getComputedStyle(btn); *// const com todos os dados de estilo do botão*

 for(*const* style in btnStyles) { *// para cada estilo*

   console.log(**`**${style}: ${btnStyles[style]}**`**); *// mostramos o nome do estilo + o valor do estilo*

**13.0 Regular Expression**

Regexp ou Regex são expressões utilizadas para realizarmos buscas / substituições de padrões em strings. Os padrões devem ser colocados entre //. Geralmente vamos utilizá-las nos métodos .replace() e .split().

*const* frase = 'JavaScript'

*const* fraseNova = frase.replace(/a/, 'b')

console.log(fraseNova) *//JbvaScript*

*Praticamente todas as linguagens possuem uma implementação de regexp*

**13.0.a Literal**

Utilizar um caracter literal irá realizar uma busca específica deste caracter.

*// Procura: J seguido de a, v e a*

*const* regexp = /Java/;

*const* nova = 'JavaScript'.replace(regexp, 'Type');

console.log(nova)*// TypeScript // substitui java por type // não precisa ter o mesmo num. de letras*

**13.0.b Flag: g**

As flags irão modificar como a expressão é interpretada. Uma das mais utilizadas é a g, que significa global, ou seja, retorne todos os resultados que estiverem dentro do padrão e não apenas o primeiro. A flag deve ser colocada no final da expressão.

*const* regexp = /a/g; *// seleciona todos a*

*const* nova = 'JavaScript'.replace(regexp, 'i'); *//sibstitui por i*

console.log(nova)*// JiviScript*

**13.0.c Flag g + i**

Com o i informamos que devem ser ignoradas as diferenças entre maiúsculas e minúsculas. Isso significa que /a/ irá buscar por a e A.

*// Procura: Todo PE, Pe, pE e pe*

*const* regexp = /Pe/gi;

console.log('Perdeu perdido'.replace(regexp, 'Ba'))*// Bardeu Bardido*

**13.0.d Character class []**

Se colocarmos os caracteres entre colchetes, estamos definindo uma classe. /[ab]/ irá procurar por a ou por b.

*// Procura: Todo a, A, i, I*

*const* regexp = /[ai]/gi;

'JavaScript'.replace(regexp, 'u');

*// JuvuScrupt*

**13.0.e Characteres especiais**

Podemos utilizar caracteres que não são alfanuméricos dentro da classe. Mas fique atento, pois existem diversos casos especiais para os mesmos.

*// Procura: - ou . ou [*

*const* regexp = /[-.[]/g;

console.log('111.222-333-4[4'.replace(regexp, ''))*// 11122233344*

**13.0.f um ou outro**

*// Procura: B, seguido de r, a*

*// seguido de s ou z, seguido de i, l*

*const* regexp = /Bra[sz]il/g;

'Brasil é com z: Brazil'.replace(regexp, 'Prazer');

*// Prazer é com z: Prazer*

**13.0.g buscando de a a z**

O traço - dentro de [] pode servir para definirmos um alcance. [A-Z] irá buscar os caracteres de A à Z. [0-9] busca de 0 à 9. A tabela UNICODE é utilizada como referência para definir os caracteres dentro do alcance.

*// Busca por itens de a à z*

*const* regexp = /[a-z]/g;

'JavaScript é a linguagem.'.replace(regexp, '0');

*// J000S00000 é 0 000000000.*

*// Busca por itens de a à z e A à Z*

*const* regexp = /[a-zA-Z]/g;

'JavaScript é a linguagem.'.replace(regexp, '1');

*// 1111111111 é 1 111111111.*

*// Busca por números de 0 à 9*

*const* regexpNumero = /[0-9]/g;

'123.333.333-33'.replace(regexpNumero, 'X');

*// XXX.XXX.XXX-XX*

**13.0.h Negar**

*// Procura: tudo que não estiver entre a e z minusculo*

*const* regexp = /[^a-z]/g;

'Brasil é com z: Brazil'.replace(regexp, ' ');

*// rasil   com z   razil*

**13.0.i Caractere especial ponto**

O ponto . irá selecionar qualquer caracter, menos quebras de linha.

*// Procura: todos os caracteres menos quebra de linha*

*const* regexp = /./g;

'JavaScript é a linguagem.'.replace(regexp, '0');

*// 0000000000000000000000000*

**13.0.j Escapar caracteres especiais**

Caracteres especiais como o ponto ., podem ser escapados utilizando a barra \, assim este não terá mais a sua função especial e será tratado como literal. Lista de caracteres especiais: +\*?^$\.[]{}()|/

*// Procura: todos os pontos*

*const* regexp = /\./g;

*const* regexpAlternativa = /[.]/g;

'999.222.222.11'.replace(regexp, '-');

*// 999-222-222-11*

**13.0.k Word**

O \w irá selecionar qualquer caracter alfanumérico e o underline. É a mesma coisa que [A-Za-z0-9\_].

*// Procura: todos os alfanuméricos*

*const* regexp = /\w/g;

'Guarda-chuva R$ 23,00.'.replace(regexp, '-');

*// ------------ -$ --,--.*

**13.0.l Not Word**

O \W irá selecionar tudo o que não for caracter alfanumérico e o underline. É a mesma coisa que [^A-Za-z0-9\_].

*// Procura: o que não for caracter alfanuméricos*

*const* regexp = /\W/g;

'Guarda-chuva R$ 23,00.'.replace(regexp, '-');

*// Guarda-chuva-R--23-00-*

**13.0.m Digit**O \d irá selecionar qualquer dígito. É a mesma coisa que [0-9].

*// Procura: todos os dígitos*

*const* regexp = /\d/g;

'+55 (21) 2222-2222'.replace(regexp, 'X');

*// +XX (XX) XXXX-XXXX.*

**13.0.n Not Digit**O \D irá selecionar tudo que não for dígito. É a mesma coisa que [^0-9].

*// Procura: o que não for dígito*

*const* regexp = /\D/g;

'+55 (21) 2222-2222'.replace(regexp, '');

*// 552122222222*

**13.0.o White space**

O \s irá selecionar qualquer espaço em branco, isso inclui espaços, tabs, quebra de linhas.

*// Procura: espaços em branco*

*const* regexp = /\s/g;

'+55 (21) 2222-  2222  '.replace(regexp, '');

*// +55(21)2222-2222*

Caso queira selecionar tudo, pode utilizar   
*const* regexp = /\S\s/g;

**13.0.p Not White space**

O \S irá selecionar qualquer caracter que não for espaço em branco.

*// Procura: o que não for espaço em branco*

*const* regexp = /\S/g;

'+55 (21) 2222-  2222  '.replace(regexp, '');

*// XXX XXXX XXXXX  XXXX*

**13.0.q Quantificador min e max**

Podemos informar o min e max do quantificador /a{2,4}/ vai selecionar quando aparecer a duas vezes ou até 4 vezes. /a{2,}/ irá selecionar quando se repetir duas ou mais vezes.

*// Procura: dígitos seguidos de 2 à 3*

*const* regexp = /\d{2,3}/g; *// quando digito se repete duas ou 3 vezes*

'222.333.222.42'.replace(regexp, 'X');

*// X.X.X.X*

*// Procura: letras seguidos com 1 caracter ou mais*

*const* regexpLetras = /\w{1,}/g; *// quando a letra é apresentada 1 ou mais vezes*

'A melhor linguagem é JavaScript'.replace(regexpLetras, 'X');

*// X X X é X*

**13.0.r Mais +**

O sinal de + significa que devemos selecionar quando existir pelo menos uma ou mais ocorrências.

*// Procura: dígitos em ocorrência de um ou mais*

*const* regexp = /\d+/g;

'222.333.222.42'.replace(regexp, 'X');

*// X.X.X.X*

*// Procura: Começa com d, seguido por uma ou mais letras.*

*const* regexpLetras = /d\w+/g;

'Dígitos, dados, desenhos, Dito, d'.replace(regexpLetras, 'X');

*// Dígitos, X, X, Dito, d*

13.0.s Asterisco \*  
O sinal \* significa que devemos selecionar quando existir 0 ou mais ocorrências.

*// Procura: Começa com d, seguido por zero ou mais letras.*

*const* regexp = /d\w\*/g;

'Dígitos, dados, desenhos, Dito, d'.replace(regexp, 'X');

*// Dígitos, X, X, Dito, X*

**13.0.t Opcional ?**

O sinal de ? informa que o caractere anterior pode ser opcional;

*// Procura: Por regex com p opcional*

*const* regexp = /regexp?/g;

'Qual é o certo, regexp ou regex?'.replace(regexp, 'Regular Expression');

*// Qual é o certo, Regular Expression ou Regular Expression?*

**13.0.u Alternado |**

O sinal | irá selecionar um ou outro. java|php

*// Procura: java ou php (case insensitive)*

*const* regexp = /java|php/gi;

'PHP e Java são linguagens diferentes'.replace(regexp, 'X');

*// X e X são linguagens diferentes*

**1.3.0.v Word Boundary**

O sinal \b irá indicar que pretendemos fazer uma seleção que deve ter início e fim de não caracteres \w.

*// Procura: java (case insensitive)*

*const* regexp = /java/gi;

'Java não é JavaScript.'.replace(regexp, 'X');

*// X não é XScript.*

*// Procura: java (case insensitive)*

*const* regexpBoundary = /\bjava\b/gi;

'Java não é JavaScript.'.replace(regexpBoundary, 'X');

*// X não é JavaScript.*

*// Procura: Dígitos em sequência, que estejam isolados*

*const* regexpDigito = /\b\d+\b/gi;

'O Restaurante25 na Rua 3, custa R$ 32,00'.replace(regexDigito, 'X');

*// O Restaurante25 na Rua X, custa R$ X,X*

'11\_22 33-44 55é66 77e88'.replace(regexpDigito, 'X');

*// 11\_22 X-X XéX 77e88*

**13.o.x Anchor beginning ^**

Com ^ é possível informar que a busca deve se iniciar no inicio da linha

*// Procura: sequência de alfanuméricos*

*// no início da linha.*

*const* regexp = /^\w+/g;

**`**andre@origamid.com

contato@origamid.com**`**.replace(regexp, 'X');

*// X@origamid.com*

*// contato@origamid.com*

**13.o.y Anchor End**

Com $ é possível informar que a busva deve ser iniciada no fim da linha

*// Procura: sequência de alfanuméricos*

*// no final da linha.*

*const* regexp = /\w+$/g;

**`**andre@origamid.com

contato@origamid.com**`**.replace(regexp, 'X');

*// andre@origamid.com*

*// contato@origamid.X*

**13.o.z Flag M**

Com a flag m de multiline, podemos informar que a busca de início ^ e final $ de linha devem ocorrer em todas as linhas disponíveis.

*// Procura: sequência de alfanuméricos*

*// no final da linha.*

*const* regexp = /\w+$/gm;

**`**andre@origamid.com

contato@origamid.com**`**.replace(regexp, 'X');

*// andre@origamid.X*

*// contato@origamid.X*

*// Procura: sequência de alfanuméricos*

*// no início da linha.*

*const* regexp = /^\w+/gm;

**`**andre@origamid.com

contato@origamid.com**`**.replace(regexp, 'X');

*// X@origamid.com*

*// X@origamid.com*

**13.o.a2 line feed /n**

O \n irá selecionar o final de uma linha, quando criamos uma nova.

*const* regexp = /\n/g;

**`**andre@origamid.com\ncontato@origamid.com**`**.replace(regexp, '---');

*// andre@origamid.com---contato@origamid.com*

**`**andre@origamid.com

contato@origamid.com**`**.replace(regexp, 'X');

*// andre@origamid.com---contato@origamid.com*

**13.0.b2 Unicode**

O \u irá selecionar o respectivo caracter unicode, de acordo com o código passado em \uXXXX. Ex: \u0040 seleciona o @.

*// Procura: @ ou ©*

*const* regexp = /\u0040|\u00A9/g;

'andre@origamid.com ©'.replace(regexp, '---');

*// andre---origamid.com ---*

**13.1 Seleção Regular Expression**

**13.1.a Referencia a seleção**

É possível utilizarmos o $& durante o momento da substituição para fazermos uma referência à seleção.

*// Procura: Java*

*const* regexp = /Java/g;

'PHP e Java são linguagens diferentes'.replace(regexp, '--$&Script');

*// PHP e --JavaScript são linguagens diferentes*

*// $& será igual à Java*

**13.1.b Grupos de captura**

É possível definirmos diferentes grupos de captura, que poderão ser referenciados durante a substituição. Basta envolvermos um grupo entre () parênteses. A referência se cada grupo será feita com $n, sendo o primeiro $1.

*// Procura: sequência alfanumérica, seguida*

*// de @, seguido de alfanumérico ou .*

*const* regexp = /(\w+)@[\w.]+/g;

'andre@email.com.br'.replace(regexp, '$1@gmail.com');

*// andre@gmail.com*

Podemos definir quantos grupos de captura quisermos.

*// Procura: sequência alfanumérica, seguida*

*// de , seguido espaço de sequência alfanumérica.*

*const* regexp = /(\w+),\s(\w+)/g;

'Rafael, Andre'.replace(regexp, '$2 $1');

*// Andre Rafael*

**13.1.c Caracteres com repetição**

Com os símbolos () de captura, também podemos indicar uma sequencia de caracteres que queremos repetição

*// Procura: qualquer sequência de ta*

*const* regexp = /(ta)+/gi;

'Tatata, tata, ta'.replace(regexp, 'Pá');

*// Pá, Pá, Pá*

Também conseguimos ignorar a captura

Com ?:

*// Procura: qualquer sequência de ta*

*const* regexp = /(?:ta)+/gi;

'Tatata, tata, ta'.replace(regexp, 'Pá');

*// Pá, Pá, Pá*

Irá selecionar, mas não dará um grupo

**13.2.d Positive e negative look ahead**

Positive

Faz a seleção dos itens que possuírem o padrão dentro de (?=) à sua frente. Apesar de utilizar () parênteses o positive lookahead não captura grupo.

*// Procura: dígitos em sequência, que*

*// possuírem px, sem selecionar o px.*

*const* regexp = /\d(?=px)/g;

'2em, 4px, 5%, 2px, 1px'.replace(regexp, 'X');

*// 2em, Xpx, 5%, Xpx, Xpx*

Negative

Faz a seleção dos itens não possuírem o padrão dentro de (?!) à sua frente.

*// Procura: dígitos que não possuírem px*

*// sem selecionar o restante.*

*const* regexp = /\d(?!px)/g;

'2em, 4px, 5%, 5px, 1px'.replace(regexp, 'X');

*// Xem, 4px, X%, 5px, 1px*

**13.2.e Seleção de CEP**

*const* regexpCEP = /\d{5}[-\s]?\d{3}/g;

*///\d{5} começa com 5 digitos*

*// [-\s]? // pode ter traço ou espaço*

*//\d{3} // seguido se 3 digitos*

*///g; // seleciona geral*

*const* ceps = [

  '87114-655',

  '87114 655',

  '87114655'

];

for(cep of ceps) {

  console.log(cep, cep.match(regexpCEP));

}

**13.2.f Seleção CPF e CNPJ**

*const* cpfs = [

  '000.000.000-00',

  '000-000-000-00',

  '000.000.000.00',

  '000000000-00',

  '00000000000'

];

for(cpf of cpfs) {

  console.log(cpf, cpf.match(regexpCPF));

}

*const* regexpCNPJ = /\d{2}[-.]?(?:\d{3}[-.]?){2}[-\/]?\d{4}[-.]?\d{2}/g;

*/\**

*/\d{2}[-.]? começa com dois digitos com - e . opcional*

*(?:\d{3}[-.]?){2} seguido de 3 digitos com - e . opcional, repete 2 vezes e não forma grupo*

*[-\/]? barra e traço opcional*

*\d{4}[-.]? seguido de 4 digitos com - e . opcional*

*\d{2}/g; // finaliza c 2 digitos*

*\*/*

*const* cnpjs = [

  '00.000.000/0000-00',

  '00000000000000',

  '00-000-000-0000-00',

  '00.000.000/000000',

  '00.000.000.000000',

  '00.000.000.0000.00',

];

for(cnpj of cnpjs) {

  console.log(cnpj, cnpj.match(regexpCNPJ));

}

**13.2.d Seleção telefone**

*const* regexpTELEFONE = /(?:\+?55\s?)?(?:\(?\d{2}\)?[-\s]?)?\d{4,5}[-\s]?\d{4}/g;

*/\**

*/(?:\+?55\s?)? // pode ou não começar com + 55 ou espaço, ou nada disso e não forma grupo*

*(?:\(?\d{2}\)?[-\s]?)? // seguido de pode ou não ter ( seguido de 2 digitos e pode ou n ter ) que pode ou n ter espaço, ou nada disso e não forma grupo*

*\d{4,5}[-\s]? // seguido de 4 ou 5 digitos, e pode ou não ter espaço ou traço*

*\d{4}/g; // seguido de 4 digitos*

*\*/*

*const* telefones = [

  '+55 11 98888-8888',

  '+55 11 98888 8888',

  '+55 11 988888888',

  '+55 11988888888',

  '+5511988888888',

  '5511988888888',

  '11 98888-8888',

  '11 98888 8888',

  '(11) 98888 8888',

  '(11) 98888-8888',

  '11-98888-8888',

  '11 98888 8888',

  '11988888888',

  '11988888888',

  '988888888',

  '(11)988888888',

  '98888 8888',

  '8888 8888'

];

**13.2.e Seleção email**

*const* regexpEMAIL = /[\w.-]+@[\w-]+\.[\w-.]+/gi;

*/\**

*/[\w.-]+ sequencia de caractere com . ou -*

*@[\w-]+ seguido de arroba com sequencia com . ou -*

*\.[\w-.]+/gi; // seguido de ponto mais sequencia com . ou - com case insensitive*

*\*/*

*const* emails = [

  'email@email.com',

  'email@email.com.org',

  'email-email@email.com',

  'email\_email@email.com',

  'email.email23@email.com.br',

  'email.email23@empresa-sua.com.br',

  'c@contato.cc',

];

**13.2.f Selecionando tag**

*const* regexpTAG = /<\/?[\w\s="']+\/?>/gi;

*/\**

*/<\/? // seleciona < e / é opcional*

*[\w\s="']+ // seleciona sequencia de caractere, espaço " e '*

*\/? // barra de fechamento é opcional*

*>/gi; //traço de fechamento*

*\*/*

*const* tags = [

  '<div>Isso é uma div</div>',

  '<div class="ativa">Essa está ativa</div>',

  '<img src="imagem" />',

  '<img src="imagem">',

  '<ul class="ativa">',

  '<li>Essa está ativa</li>',

  '</ul>'

];

for(tag of tags) {

  console.log(tag, tag.match(regexpTAG));

}

**13.2.g look behind**

Não indicado, não é suportado

*const* regexpTAG = /(?<=<\/?)[\w]+/gi;

*/\**

*/(?<=<\/?) se atras tiver < com ; opcional*

*[\w]+/gi; // seguido de caracteres*

*\*/*

*const* tags = [

  '<div>Isso é uma div</div>',

  '<div class="ativa">Essa está ativa</div>',

  '<img src="imagem" />',

  '<img src="imagem">',

  '<ul class="ativa">',

  '<li>Essa está ativa</li>',

  '</ul>'

];

for(tag of tags) {

  console.log(tag, tag.match(regexpTAG));

}

**13.3 RegExp Constructor**

Toda regular expression. É criada com o constructor RegExp() e herda propriedades e métodos   
Existe diferença entre uma RegExp criada com uma variável e uma criada com o argumento de RegExmp

*// 1° Criada diretamente em uma string*

*const* regexp = /\w+/gi; *//*

*// 2° Criada como argumento de uma RegularExpression*

*// 3° String*

*// e devemos utilizar \\ para meta characters, pois é necessário*

*// escapar a \ especial. As Flags são o segundo argumento*

*// Se passarmos uma string, não precisamos dos / /*

*// Seria util quando o usuario for digitar uma RegExmp, pois virá como string*

*const* regexpObj1 = new RegExp('\\w+', 'gi');

*// 4° Argumento*

*const* regexpObj2 = new RegExp(/\w+/, 'gi');

'JavaScript Linguagem 101'.replace(regexpObj1, 'X');

*// X X X*

*// Exemplo complexo:*

*//argumento*

*const* regexpTELEFONE1 = /(?:\+?55\s?)?(?:\(?\d{2}\)?[-\s]?)?\d{4,5}[-\s]?\d{4}/g;

*//string*

*const* regexpTELEFONE2 = new RegExp('(?:\\+?55\\s?)?(?:\\(?\\d{2}\\)?[-\\s]?)?\\d{4,5}[-\\s]?\\d{4}', 'g');

*// Metodo em const*

*const* regexp = /\w+/gi

*// Metodo em string*

*const* regexp1 = new RegExp('\\w+', 'gi')

*const* frase = "JavaScript Linguagem 101";

console.log(frase.replace(regexp, 'X')) *// X X X*

**13.3.a Propriedades**

*const* regexp = /\w+/gi;

regexp.flags; *// 'gi'*

regexp.global; *// true*

regexp.ignoreCase; *// true*

regexp.multiline; *// false*

regexp.source; *// '\w+'*

**13.3.b RegExp.Test**

O método test() verifica se existe ou não uma ocorrência da busca. Se existir ele retorna true. A próxima vez que chamarmos o mesmo, ele irá começar do index em que parou no último true.

*const* regexp = /Java/g;

*const* frase = 'JavaScript e Java';

regexp.lastIndex; *// 0*

regexp.test(frase); *// true*

regexp.lastIndex; *// 4*

regexp.test(frase); *// true*

regexp.lastIndex; *// 17*

regexp.test(frase); *// false*

regexp.lastIndex; *// 0*

regexp.test(frase); *// true (Reinicia*

regexp.lastIndex;  *// 4*

**13.3.c RegExp.test em loop**

Podemos utilizer o while para fazer o loop, e por ex indicar em qual índice está o procurado

*const* regexp = /Java/g;

*const* frase = "JavaScript Linguagem 101 Java";

*let* i = 1

while(regexp.test(frase)) {

  console.log(i++, regexp.lastIndex)

}

*// 1  4 // primeiro loop local 4 terminou*

*// 2  29 // segundo loop local 29 terminou*

**13.3.d RegExmp.Exec**

exec() diferente do test(), irá retornar uma Array com mais informações do que apenas um valor booleano.

*const* frase = "JavaScript, TypeScript e CoffeeScript";

*const* regexp = /\w{2,}/g;

*//exec não podemos rodar diretamente pois entra em loop infinito com null*

*//então colocamos dentro de uma variavel*

*let* regExmpresult;

while((regExmpresult = regexp.exec(frase)) !== null) { *//regExmpresult recebe o exec, e enquanto for diferente de null*

console.log(regExmpresult[0]) *// exibibos o selecionado*

**13.3.e RegExmp.Match**

O match() é um método de strings que pode receber como argumento uma Regexp. Existe uma diferença de resultado quando utilizamos a flag g ou não.

*const* regexp = /\w{2,}/g;

*const* regexpSemG = /\w{2,}/;

*const* frase = 'JavaScript, TypeScript e CoffeeScript';

console.log(frase.match(regexp))

*// ['JavaScript', 'TypeScript', 'CoffeeScript']*

console.log(frase.match(regexpSemG))

*// ["JavaScript", index: 0, input: "JavaScript,*

*// TypeScript e CoffeeScript", groups: undefined]*

*Caso não tiver match retorna null*

**13.3.f RegExmp e Split**

Utiizando o método split na string, com o regexp, ele irá remover o que der match, e fazer uma array

*const* frase = 'JavaScript, TypeScript, CoffeeScript';

*const* regexp = /Script/g

console.log(frase.split(regexp)) *// (4) ['Java', ', Type', ', Coffee', '']*

**13.3.g regexp Replace**

O método replace() é o mais interessante por permitir a utilização de funções de callback para cada match que ele der com a Regexp.

*// const literal com uma ul*

*const* tags = **`**

<ul>

  <li>Item 1</li>

  <li>Item 2</li>

</ul>

**`**;

*const* regExp = /(?<=<\/?)\w+/g *// com essa regexp vamos selecionar com lookback o que tiver entre setas*

*const* result = tags.replace(regExp, 'div'); *// utilizando a diz que criamos substituimos por div*

console.log(result)

*// <div>*

*//   <div>Item 1</div>*

*//   <div>Item 2</div>*

*// <div>*

Replace com Referencia

*const* regExp2 = /<li/g;

console.log(tags.replace(regExp2, '$& class=\'active\''))

*// <div>*

*//   <div class='active'>Item 1</div>*

*//   <div class='active'>Item 2</div>*

*// <div>*

*const* emails = **`**

empresa@email.com

contato@email.com

suporte@email.com

**`**;

*const* regexp3 = /(\w+@)[\w.]+/g *//primeiro grupo, nome + @, segundo grupo sequencia e ponto*

console.log(emails.replace(regexp3, "$1gmail.com")) *// substitui por referencia grupo 1 + string*

**13.3.h Callback de ragexp**

Para substituições mais complexas, podemos utilizar um callback como segundo argumento do replace. O valor do return será o que irá substituir cada match.

*const* emails = **`**joao@homail.com.br

marta@ggmail.com.br

bruna@oulook.com.br

gearqq@hotmail.com**`**;

*const* regexp3 = /(\w+@)([\w.]+)/g

*const* resultado = emails.replace(regexp3, *function*(...*args*) {

  console.log(args)

  if(args[2] === 'homail.com.br') {

    return args[1] + 'hotmail.com'

  } else if(args[2] === 'ggmail.com.br') {

    return args[1] + 'gmail.com'

  }else if(args[2] === 'oulook.com.br') {

    return args[1] + 'outlook.com.br'

  } else {

    return args[0]

  }

})

console.log(resultado)

*/\**

*Array de argumentos*

*0:"joao@homail.com.br"*

*1:"joao@"*

*2:"homail.com.br"*

*3:0*

*\*/*

**13.3.i Limpar CPF**

Grupo de funções para limpar, interar e inserir numero editado

*// Função para selecionar uma lista de CPF no html e retornar editado*

*const* cpfsLista = document.querySelectorAll(".cpf li"); *// 1° Selecionamos a li que tem a lista*

*// em uma function expression utilizamos o spread em uma arrow function para passar o map em cada li rretornando o innertext*

*const* elementsInnerText = ([...*elements*]) => {

  return elements.map((*element*) => element.innerText);

};

*// Utilizamos regular expression em uma arrow function para remover tudo que não é digito e substituir por nada*

*const* limparCpf = (*cpf*) => {

  return cpf.replace(/\D/g, "");

};

*// regular expression para separar os numeros em grupo, e colocar pontuações corretas*

*const* construitCPF = (*cpf*) => {

  return cpf.replace(/(\d{3})(\d{3})(\d{3})(\d{2})/g, "$1.$2.$3-$4");

};

*// função que ativa a função de limpar e logo em seguida a de construir com o map*

*const* formatarCpfs = (*cpfs*) => {

  return cpfs.map(limparCpf).map(construitCPF);

};

*// função que ativa as outras 3, e envia para o html*

*const* subCpf = (*cpfsElements*) => {

*const* cpfs = elementsInnerText(cpfsElements); *// recebe a array a FE elementsInnerText*

*const* cpfsFormatados = formatarCpfs(cpfs); *// ativa a função que ativara as outras duas para limpar e construir o cpf*

  cpfsElements.forEach((*element*, *index*) => {

    element.innerText = cpfsFormatados[index];*// for each que por index envia para o innertext o cpf formatado*

  });

};

subCpf(cpfsLista); *// ativando a função que irá ativar as outras, e incluir no html*

**13.4 Validação, limpeza e verificação visual de CPF**

import ValidarCpf from "./validarCpf.js";

*// Para iniciar, puxamos a lista que tem o cpf para uma const*

*// e ativamos na constructor class o metodo iniciar*

*const* cpf = document.querySelector("#cpf");

*const* validarCpf = new ValidarCpf(cpf).iniciar();

export default *class* validarCpf {

*constructor*(*element*) {

    this.element = element;*//recebe a lista de cpf*

  }

  limpar(*cpf*) {

    return cpf.replace(/\D/g, "");*// 12° retorna apenas digitos*

  }

  construir(*cpf*) {

    return cpf.replace(/(\d{3})(\d{3})(\d{3})(\d{2})/g, "$1.$2.$3-$4"); *//14° Aciciona traços com regexp*

  }

  formatar(*cpf*) {*//10° caso deja valido*

*const* cpfLimpo = this.limpar(cpf);*//11° passa pelo metodo de linpar*

    return this.construir(cpfLimpo);*//13° e retorna passado pelo construir*

  }

  validar(*cpf*) {  *//7° metodo que ira validar e retornar true ou false se o metodo de cpf está correto*

*const* matchCPF = cpf.match(/(?:\d{3}[-.\s]?){3}\d{2}/g)*//*

    return(matchCPF && matchCPF[0] === cpf) *// caso o cpf corresponder com o regexp, irá retornar true*

  }

  validarNaMudança(*cpfElement*) { *//5° metodo que ira limpar,validar adicionar e remover classe de erro*

    if(this.validar(cpfElement.value)) { *//6° acionando o metodo de validação com o this // 8° caso esteja correto if, caso não false*

      cpfElement.value = this.formatar(cpfElement.value) *//9° caso true, entra para formatar.. apenas o value*

      cpfElement.classList.add('valido')*//recebe classe de valido*

      cpfElement.classList.remove('erro') *// remove classe de invalido*

      cpfElement.nextElementSibling.classList.remove('ativar') *// remove elemento incluido para anunciar erro*

    } else {

      cpfElement.classList.add('erro') *//caso falso adiciona classe de erro*

      cpfElement.classList.remove('valido') *// remove classe de valido*

      cpfElement.nextElementSibling.classList.add('ativar') *// adiciona elemento de invalido*

    }

  }

  adicionarEvento() {*//3° evento que verifica se ouve mudanca em this element*

    this.element.addEventListener("change", () => {

      this.validarNaMudança(this.element);*//4°caso haja aciona o metodo de validação*

    });

  }

  adicionarErroSpan() {

*const* errorElement = document.createElement('span') *// cria um span que recebera classe de atvo, por padrão é display none*

    errorElement.classList.add('errotext') *// display none*

    errorElement.innerText = 'CPF Invalido'*//texto da caixa*

    this.element.parentElement.insertBefore(errorElement, this.element.nextElementSibling)*// inserindo antes do proximo filho, o elemento criado errorElement*

  }

  iniciar() { *//1° inicia o plugin de validação*

    this.adicionarEvento() *// 2° com o this(input selecionada) acionamos o metodo de add eventos de mudança*

    this.adicionarErroSpan()*// ativando metodo que adiciona um elemento por JS para mensagem de erro*

    return this

  }

}

**14.0 Automação front end**

Utilizando o gitbash ou visual studio com o gitbash

**14.1 Linha de comando – CLI**

Interagir com o Computador através de texto

Comando começando com $ (Unix) ou > (Win)

UNIX (Mac e Linux) vs Windows

Bash e Zsh (Unix) | CMD e PowerShell (Windows)

<https://ss64.com>

**14.0.a Bash**

Programa Git-Bash

## MOVIMENTAR (BASH)

* $ cd sites

vai para a pasta sites

* $ cd ..

vai para a pasta anterior

* $ cd ~/Desktop/sites

vai para a pasta sites dentro de Desktop

* $ cd ~

vai para o diretório principal do sistema / home

* $ clear

Limpa a tela

## LISTAR E CRIAR (BASH)

* $ ls

Lista diretórios e arquivos

* $ ls -all

Lista diretórios, arquivos, invisíveis e detalhes

* $ mkdir site

Cria o diretório site

* $ touch index.html

Cria o arquivo index.html

## REMOVER (BASH)

* $ rm index.html

Remove o arquivo index.html

* $ rm -r site

Remove o diretório site e todos os arquivos dentro dele

* seta para cima / baixo

Acessa o comando anterior

* tab

Auto-completa o comando

**14.0.b instalando novo pacote npm**

Ao criar um novo projeto, é de importância criar o npm init com o bash, assim é criado um arquivo, que irá listar todos os pacotes necessários para rodar aquele código,

Assim ao abrir um grupo de arquivos que já tenha o package.json com os pacotes dependentes, podemos dar o comando $ npm install que será instalado os necessários

**14.0.c comandos para instalação de pacotes**

## INSTALAR PACOTES

* $ npm install lodash

Instala o pacote lodash no diretório atual da linha de comando

* $ npm install eslint -g

Instala o eslint globalmente(no computador, não na pasta) (No mac é necessário o sudo para instalar globalmente.)

* $ npm update lodash

Atualiza o pacote lodash

* $ npm uninstall lodash

Desinstala o pacote lodash

## PACKAGE.JSON

* package.json

Arquivo local com as configurações e dependências de pacotes NPM

* $ npm init

Inicia uma nova configuração local do npm.

* $ npm install

Instala todas as dependências listadas no arquivo package.json

**14.0.c Eslint**

**ESLINT**

* Evitar problemas

Indica a existência de possíveis padrões problemáticos no código

* Definir padrões

Padronizar e manter consistência entre diferentes códigos JavaScript

* [https://eslint.org](https://eslint.org/)

**INSTALAR ESLINST**

* $ npm install eslint -g

Instalar o eslint globalmente.

* $ npm init

Iniciar repositório NPM no local.

* $ eslint --init

Use Popular > Airbnb > (N) Enter > JSON > (Y) Enter

* Instalar extensão ESLint do VS Code

**14.1 Webpack**

Bundler

Agrupa / processa diversos arquivos e otimiza os mesmos.

Altamente configurável, Por isso é complexo.

<https://webpack.js.org/>

1. Instalar o eslint

$ npm install eslint

1. Iniciar o npm n local

$npm int

1. Iniciar o eslint

$ eslint –init

    "extends": ["airbnb-base", "prettier"],

Webpack

1. Iniciar como dev dependencies com o comando

$ npm install --save-dev webpack webpack-cli

  "devDependencies": {

    "eslint": "^8.40.0",

    "eslint-config-airbnb-base": "^15.0.0",

    "eslint-plugin-import": "^2.27.5",

    "webpack": "^5.82.0",

    "webpack-cli": "^5.1.1"

  },

1. Acionando o webpack

Caminho do original + para onde vai o novo

$ npx webpack ./js/script.js -o ./main.js

$ npx webpack ./js/script.js -o ./main.js --watch

1. Altere o arquivo principal do html para o criado pelo webpack

  <script src="/main.js/main.js"></script>

1. Deixe rodando em package json um watch, para que seja atualizado com o script do dev constantemente

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1"

  },

\/

{

  "name": "1004-webpack",

  "version": "1.0.0",

  "main": "index.js",

  "devDependencies": {

    "eslint": "^8.40.0",

    "eslint-config-airbnb-base": "^15.0.0",

    "eslint-plugin-import": "^2.27.5",

    "webpack": "^5.82.0",

    "webpack-cli": "^5.1.1"

  },

  "scripts": {

    "build": "webpack --mode production ./js/script.js -o ./main.js",

    "dev": "webpack --mode development --watch ./js/script.js -o ./main.js"

  },

  "author": "",

  "license": "ISC",

  "description": ""

}

Dessa forma irá ter o comando build, que será para subir o site

E o comando Dev, que tem o watch, e irá ficar observado toda a mudança que tiver na construção

Assim, quando for finalizado o código, rode ele no modo build para que o arquivo main fique da forma ideal

1. Scripts externos

Podemos facilmente importar scripts externos instalando os mesmos através do NPM e utilizando o Webpack para fazer o bundler final.

$ npm install jquery lodash

  "dependencies": {

    "jquery": "^3.6.4",

    "lodash": "^4.17.21"

  }

Exemplos de $ e \_

import $ from "jquery";

import \_, { difference } from "lodash";

$('nav').hide()

*const* diference =\_.difference(['banana', 'uva'], ['banana', 'morango'])

console.log(diference)

**14.1 Babel**

Compilador

Transforma código novo em código antigo. Ex: const nome = 'Andre'; vira var nome = 'Andre';.

Instalando babel, na linha de comando bash utilizar o cod;

$ npm install --save-dev @babel/core @babel/preset-env @babel/plugin-transform-runtime babel-loader

Crie um novo arquivo com o nome exato; webpack.config.js

E com o seguinte código,

*const* path = require('path');

module.exports = {

  entry: './js/script.js',

  output: {

    path: path.resolve(\_\_dirname, './'),

    filename: 'main.js',

  },

  module: {

    rules: [

      {

        test: /\.js$/,

        exclude: /node\_modules/,

        use: {

          loader: 'babel-loader',

          options: {

            presets: ['@babel/preset-env'],

            plugins: ['@babel/plugin-transform-runtime'],

          },

        },

      },

    ],

  },

};

E em package json altere o código de dev e build para

  "scripts": {

    "build": "webpack --mode production",

    "dev": "webpack --mode development --watch"

  },

14.2 Refatorando

Para iniciar, criar um brench com: $ git checkout -b (nome do galho desejado)  
e run deve para ser executado com watch

Após realizar as mudanças, fechar o watch com cntrl + c

Npm Run build

Realizar commit com o nome da mudança   
$ git commit -m 'ScrollSuave transformado em class'

Mudar para o main  
$ git checkout main

E dar um pull   
$ git pull

Voltar para o galho que estava trabalhando

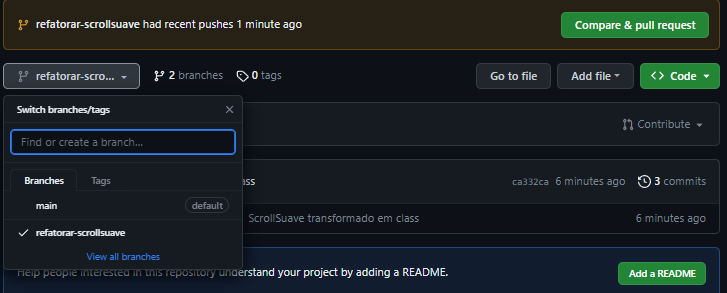
$ git checkout refatorar-scrollsuave

Dar git push para ter o comando do push

$ git push

= git push --set-upstream origin refatorar-scrollsuave

A brench deve aparecer no github para enviar ao main



Onde podemos dar continuidade, e deletar o Branch no final

E também na linha de comando, de checkout para o min e delete o brench com

$ git branch -d refatorar-scrollsuave

E finaliza com git pull

$ git pull