# JavaScript – DIO - Anotações das Aulas

## 01 – Introdução ao JavaScript

### História Evolução e Aplicações:

- Primeira versão em 1997

- Linguagem Interpretada

- Baseada em Protótipos

- Multiparadigma

- Comumente utilizada em aplicações web client-side

- Segue o padrão ECMAScript (com grande evolução em 2015)

- Aplicações: Web, Mobile, Smartwatches, Games, IoT, APIs

### Recursos Básicos:

- Manipulando um arquivo Javascript

- comentários -> // ou /\*comentário\*/ ou seleciona o texto ctrl+/

- definição de variável e constante

- Funções - foi mostrado como declarar e como chama-las

### Console:

- Executando um arquivo JS

- Foi mostrado como usar o console numa página Web para executar código.

- Foi mostrado também como instalar o NodeJS e como executar o código no terminal. (Eu já havia feito isso antes)

### Javascript em uma página Web:

Estrutura do projeto

- Mostrou a estrutura de um arquivo HTML.

Inserindo Javascript numa página HTML

- Mostrou como ligar o arquivo HTML ao CSS e ao JS

Interagindo com os elementos do DOM

- Mostrou a estrutura do DOM

- Mostrou o getElementById

### Colocando em Prática

- Criando um contador

- Criou uma página básica HTML com um número entre dois botões (+ e -). Ao clicar no + incrementa o número e no - decrementa. (Por já ter feito projetos mais complexos usando JS, optei por não fazer esse).

## 02 - SINTAXE E OPERADORES

### Operadores:

ATRIBUIÇÃO:

= x=5 x=5

+= x+=5 x=x+5

\*= x\*=5 x=x\*5

/= x/=5 x=x/5

%= x%=5 x=x%5

ARITMÉTICOS:

Além de + - \* /

\*\* -> Exponencial

% -> Módulo - Resto da divisão

++ -> Incrementar

-- -> Decrementar

COMPARAÇÃO:

== -> igual a;

=== -> mesmo valor e mesmo tipo;

!= -> diferente a;

!== -> valor e tipo diferente;

além de > < >= <=

LÓGICA:

&& -> e lógico;

|| -> ou lógico;

! -> não lógico;

CONDICIONAL (ternário) substitui o IF ELSE

(condição) ? expresão1 : expresão2

(condição) ? verdadeiro : falso

exemplo:

(i%2=0) ? console.log(i + ' é par!') : console.log(i + ' é impar!')

## 03 - VARIÁVEIS E TIPOS

### Boas Práticas:

Camel Case - nomeDaVariavel

Snake Case - nome\_da\_variavel

Kebab Case - nome-da-variavel

Pascal Case - NomeDaVariavel

Upper Case Snake Case - NOME\_DA\_VARIAVEL (CONSTANTES ou FUNÇÕES)

### VARIÁVEIS:

- var escopo global

- let escopo de bloco

- Hoisting = atribuir um valor antes de declara-la.

- camelCase

### CONSTANTES:

- SNAKE\_UPPER\_CASE

- Não faz Hoisting

- Escopo de bloco

### TIPOS:

ESTRUTURAS DE DADOS:

-JS - Tipagem Dinâmica e Fraca;

Tipos Primitivos:

* Numbers
* Strings - Textos ou frases
* Boolean

Compostos / Não Primitivos:

-Objetos - chave valor

## 04 – FUNÇÕES

## 05 - COLEÇÕES CHAVEADAS - MAPs e SETs

### MAPs

Estrutura

const myMap = new Map()

* É uma coleção de arrays no formato chave e valor
* Pode ser iterado por um loop for...of

Métodos:

myMap.set('apple', 'fruit'); // Map(1) {"apple" => "fruit"}

myMap.get(apple) // retorna fruit

myMap.delete("apple") // retorna true

myMap.get(apple) // retorna UNDEFINED

Diferença entre MAP e CHAVE

* MAP poder ter chave de qualquer tipo;
* Tem a propriedade lenght;
* São mais fáceis de iterar;
* Melhor utilizado quando o valor de uma chave é desconhecida;
* Os valores do MAP tem o mesmo tipo.

### SETs

Estrutura:

const myArray = [1, 1, 2, 2, 3, 4, 5, 6, 7, 8, 2, 1];

const mySet = new Set(myArray) // myset terá apenas[1, 2, 3, 4, 5, 6, 7, 8]

* Armazena apenas valores únicos (não pode repetir)

Métodos:

mySet.add(1); //add um item

mySet.has(1); // vê se tem o item 1 no Set e retorna true ou false

mySet.delete(1); // deleta o item 1

Diferença entre SET e ARRAY

* Em vez de lenght tem o método size
* Tem um número limitado de métodos, comparados ao array que é mais flexível.

## 06 - MAP, FILTER E REDUCE - TRABALHANDO COM ARRAYS

### MAP

Método que serve para transformar um array em outro array através de uma função.

* Exemplo: o array [1,2,3] se torna [2,4,6] pela função x=2y.
* O método MAP não modifica o array original, mas sim cria um novo como retorno.

Estrutura:

* array.map(callback, thisArg) // callback é a função a ser executada a cada item.
* callback(item,index,array) // são os valores que conseguimos acessar.
* thisArg é um argumento opcional e pode ser usado como this na função.

MAP x forEach

* array.map((item) => item \* 2); // retorno será outro array
* array.forEach((item) => item \* 2); // retorno será UNDEFINED e deveremos usar um array auxiliar para essa operação.

### FILTER

Método que filtra o array e te retorna um novo array com os items que passarem pelo filtro.

* array.filter(callback,thisArg)

callback(item,index,array) // são os valores que conseguimos acessar.

thisArg é um argumento opcional e pode ser usado como this na função.

* dado o array frutas ['maça verde','maça fuji','laranja','abacaxi']

frutas.filter((fruta) => fruta.includes('maça')) //retorno ['maça verde','maça fuji']

e o array frutas permanece intacto.

### REDUCE

É um método que executa uma operação em todos os itens de um array e retorna um resultado único.

array.reduce(callbackFn, initialValue) // callback função a ser executada a partir do acumulador; // initialValue é o valor sobre o qual o resultado final irá atuar.

const callbackFn = function(accumulador, currentValue, index, array;

## 07 - DEBUGGING E ERROR HANDLING

### TIPOS DE ERROS

**ECMA SCRIPT ERROR**

* Ocorre em tempo de execução.
* Composto por Mensagem, Nome, Linha e CallStack
* Erro de falta de ponto e virgula, varivel não declarada, etc. Erro mais comum.

**DOM EXCEPTION**

* Document Object Model - Página Web
* String com caracter inválido OU adicionar um elemento onde não é possível.

### TRATAMENTO DE ERROS

**THROW**

* dado o codigo: IF (!String) return "String Inválida"; Esse return é do tipo string.
* dado o codigo: IF (!String) THROW "String Inválida"; Esse return é do tipo ERRO e é mostrado de forma diferente no console.

**TRY...CATCH**

* TRY - Tenta executar o código ali dentro.
* CATCH - É executado caso o código do TRY tenha um erro.

Exemplo:

try {

verificaString(palavra);

}

catch(e) { // "e" seria o objeto erro retornado em algum momento no try. Aqui no catch ele pode ser tratado caso necessário.

throw e;

}

**FINALLY:**

* Esse bloco é opcional no TRY...CATCH e é usado após o catch.
* Esse bloco é executado independente do processo ter dado erro ou não.

### OBJETO ERROR

Um erro aceita os parametros message, fileName, lineNumber. Todos opcionais.

const MeuErro = new Error('Mensagem Invalida')

MeuErro.name = 'InvalidMessage';

throw MeuErro;

## 08 - JAVASCRIPT ASSÍNCRONO

### ASSÍNCRONICIDADE

**DEFINIÇÃO:**

* Que não ocorre no mesmo momento.
* JS por padrão roda de forma síncrona, uma coisa de cada vez. No cliente e no servidor.
* Ao criar processos assíncronos, podemos executar duas coisas ao mesmo tempo, e usar o servidor pra processar algo enquanto espera uma ação do cliente por exemplo.

**PROMISES:**

* Promises é um tipo de objeto usado para processamento assíncrono.
* Inicialmente, o valor de um objeto promises é desconhecido.
* Esse objeto pode ser resolvido: then() ou rejeitado: catch().
* Ela pode ter 3 estados: PENDING - FULFILLED - REJECTED

**ASYNC / AWAIT:**

Duas palavras reservadas que servem para:

* ASYNC: definir o tipo de função que será resolvida assincronamente.
* AWAIT: é executado para dar o retorno da função assíncrona.

//Estrutura do objeto assíncrono PROMISE que retorna o 'resolve' depois de 2 segundos.

ASYNC function resolvePromise() {

const myPromise = new Promise((resolve,reject) => {

window.setTimeout(() => {

resolve('Resolvida');

},2000);

})

//Esse AWAIT é executado enquanto aguarda a resposta do objeto PROMISES

//Quando é resolvida ele passa em cada um desses then

const resolved = await myPromise

.then((result) => result + ' passando pelo then') //É possível usar o try catch para esse tratamento

.then((result) => result + ' agora acabou!') //Colocando os .then no TRY e o .catch no CATCH

.catch((err) => console.log(err.message));

return resolved;

}

### APIs e FETCH

**APIs (Application Programming Interface)**

* É uma forma de intermediar resultados de duas aplicações.
* Pode ser acessada por meio de uma URL.
* JSON - JavaScript Object Notation, arquivo tipo XML que normalmente é usado na comunicação com APIs.

**FETCH**

* É um MÉTODO usado para requisitar algo de uma API.
* Devemos passar como parâmetros nesse método, a URL da API e as opções.
* Ao receber o retorno desse função devemos converter o resultado para JSON - resultado.json()
* O retorno desse método é um objeto do tipo PROMISE, então é preciso declarar o AWAIT.
* Pode ser usado para operações de Banco de Dados por exemplo: POST, GET, PUT, DELETE;

## 09 - ORIENTAÇÃO A OBJETOS

### PARADIGMAS

* Imperativo
* Pilares da Orientação a Objetos:
  + **Abstração:** tornar simples um processo complexo afim de entendimento. "generalizar e simplificar"
  + **Herança:** Objetos filhos herdam propriedades e métodos dos pais.
  + **Encapsulamento:** Cada classe tem propriedades e métodos independente do restante do código.
  + **Polimorfismo:** Objetos podem herdar a classe de um pai mas se comportar de uma forma diferente.

### PROTÓTIPOS E CLASSES

**OOJS** - Orientação a objeto no JavaScript

**PROTÓTIPOS:**

Todos os objetos no JS herdam propriedades e métodos de um protótipo mesmo que não declarado.

**CLASSES:**

* Classes não existem no JS. O que parece ser um classe no JS na verdade é um SYNTATIC SUGAR, que é um forma de ver na tela que facilita a leitura, porém, por debaixo dos panos as classes são na verdade OBJETOS.
* A Herança se dá por meio de protótipos.
* Anatomia de uma "CLASSE":
  + Construtor (criar um objeto da classe gerando as propriedades iniciais)
  + GETTER E SETTER (enviar e retornar propriedades)
  + super() (envia os parâmetros para a classe PAI)
  + Médodos específicos para cada classe.

## 11 - MANIPULANDO A D.O.M COM JavaScript

### D.O.M.

* Document Object Model.
* O DOM HTML e um padrão de como acessar e modificar os elementos HTML de uma página.
* DOM vs BOM - Browser Object Model
  + Quando abrimos um navegador temos o BOM que é o objeto que referencia o browser.
  + window é a classe mãe do BOM e tudo está dentro dessa classe.

### MÉTODOS DO D.O.M.

**Estutura do HTML:**

* Tag: cada elemento do documento. Por exemplo: <body> <p> <section> <div>
* ID: Identificador de um elemento. Um ID deve ser único no documento.
* Class: Uma classe de determinado elemento. Pode ter mais de um com o mesmo nome no documento.

**MÉTODOS:**

* document.getElementByID(titulo) -> retorna o elemento com o id solicitado.
* document.getElementsByTagName('li') -> retorna um array com os elementos da tag solicitada.
* document.getElementsByClassName('texto') -> retorna um array com os elementos da classe solicitada.
* document.querySelectorAll('.classe1 .classe2') -> retorna um array dos elementos que tenha essas duas classes.
* document.querySelectorAll('li .classe1') -> retorna um array dos elementos 'li' que seja da classe1.
* document.createElement(element) - Cria um elemento novo no HTML.
* document.removeChild(element) - Remove um elemento filho de dentro de um elemento pai.
* document.appendChild(element) - Adiciona um elemento filho dentro de um elemento pai.
* document.replaceChild(elementNEW, elementOLD) - Substitui um elemento.

### TRABALHANDO COM ESTILOS

const meuElemento = document.getElementByID('meu-elemento')

meuElemento.classList.add('novo-estilo') //Adiciona a classe novo-estilo.

meuElemento.classList.remove('classe') //remove a classe 'classe' do element.

meuElemento.classList.toggle('dark-mode') //se tiver tira, se não tiver add.

document.getElementsByTagName('p').style.color = "blue";

### TIPOS E ACIONAMENTO DE EVENTOS

* Quando algo acontece na página.
* Evento de mouse - mouseover, mouseout (mouse em cima do element ou quando sai)
* Evento de mouse - click, dbclick
* Evento de atualização - change (quando um campo mudar), load(quando a página terminar de carregar)

Como chamar esses eventos:

* No JS
  + document.getElementByID('meuBotao').addEventListner(tipo do evento, função que vai chamar)
  + document.getElementByID('meuBotao').addEventListner('click', trocaTexto)
* No HTML
  + <h1 onclick="mudatexto(this)">Clique aqui</h1>