Introdução a JavaScript

Feito por: Ruan Tiago

Tópicos a serem abordados

- Introdução ao JavaScript
- Declarações de variáveis
- Tipos de variáveis
- Operadores
- Funções
- Objeto e Array
- Métodos async
- Manipulação de elementos DOM

Introdução

JavaScript é uma linguagem de script orientada a objetos e plataforma cruzada usada para tornar as páginas da Web interativas, por exemplo, adicionando animações complexas, botões clicáveis, menus pop-ups entre outros.

JavaScript e Java são similares em algumas coisas, mas são diferentes em outras. O JavaScript não possui tipagem estática e checagem rígida de tipos como o Java, além de não poder escrever automaticamente no disco rígido.

Tipos de variáveis

Boolean: false ou true

- Ex: var a = true

null: indica um valor nulo, pelo JS ser case-sensitive Null ou NULL ou qualquer outra variação não será interpretada como propriamente null.

- Ex: var b = null

undefined: uma propriedade cujo valor é indefinido.

- Ex: var c = undefined

Number: variável do tipo numero

- Ex: var d = 14 ou até mesmo d = 3.14

String: variável do tipo texto, observe as aspas envolta no texto.

- Ex: var e = “Olá, Mundo”

Existem outros tipos de variáveis, vamos abordá-los mais tarde...

Declarações de variáveis

Existem três tipos de declarações em JavaScript.

“var” : Declara uma variável global ou local, pode ser inicializado com um valor ou não.

Exemplo: var x = 10 (tipo number)

“let” : Declara uma variável local ao escopo do bloco, podendo ser inicializado com um valor ou não.

Exemplo: let y = “10” (tipo string)

“const” : Declara uma constante de escopo de bloco, o valor atribuído a esse tipo de variável não poderá ser alterado no futuro.

Exemplo: const z = [] (tipo array)

Preparação do ambiente

Vamos praticar os conceitos vistos até agora...

Para começar vamos instalar o nosso editor de texto, VSCode, vai ser ele que iremos usar nas aulas para criar e editar nossos códigos.

Link de download: https://code.visualstudio.com

Vamos instalar algumas extensões para auxiliar na construção do código.

Essenciais: HTML CSS Support, IntelliCode.

Recomendados: IntelliCode API Usage Examples, Auto Import, Live Server

Preparação do ambiente

Para adicionar o seu arquivo javascript em seu HTML basta adicionar:

<script src="caminho para seu arquivo/nome do arquivo.js"></script>

No <head> do seu HTML

Prática 1

Declare uma variável de cada tipo abaixo.

- Boolean
- Null
- Number
- String

Agora digite: “console.log(NOME DA SUA VARIÁVEL)” para ver o resultado no console no seu navegador.

Operadores aritméticos

Adição (+): Usado para somar dois valores.

Subtração (-): Usado para subtrair o segundo valor do primeiro.

Multiplicação (*): Usado para multiplicar dois valores.

Divisão (/): Usado para dividir o primeiro valor pelo segundo.

Resto da Divisão (%): Retorna o resto da divisão do primeiro valor pelo segundo.

- Exemplo: 10 % 3 resulta em 1.

Incremento (++): Adiciona 1 ao valor da variável.

- Exemplo: let x = 5; x++;

  resulta em x = 6.

Decremento (--): Subtrai 1 do valor da variável.

- Exemplo: let y = 8; y--;

  resulta em y = 7.

Prática 2

Declare duas variáveis do tipo number, multiplique elas e armazene o resultado em uma variável, após isso, exiba-a em um “console.log()”

Operadores de comparação

Igual (==):

Verifica se dois valores são iguais, mesmo que seus tipos sejam diferentes.

Exemplo: 5 == "5" é verdadeiro.

Estritamente Igual (===):

Verifica se dois valores são iguais em valor e tipo.

Exemplo: 5 === "5" é falso.

Diferente (!=):

Verifica se dois valores são diferentes, mesmo que seus tipos sejam diferentes.

Exemplo: 5 != "3" é verdadeiro.

Estritamente Diferente (!==):

Verifica se dois valores são diferentes em valor ou tipo.

Exemplo: 5 !== "5" é verdadeiro.

Maior que (>):

Verifica se o valor à esquerda é maior que o valor à direita.

Exemplo: 10 > 5 é verdadeiro.

Maior ou igual a (>=):

Verifica se o valor à esquerda é maior ou igual ao valor à direita.

Exemplo: 10 >= 10 é verdadeiro.

Menor que (<):

Verifica se o valor à esquerda é menor que o valor à direita.

Exemplo: 5 < 10 é verdadeiro.

Menor ou igual a (<=):

Verifica se o valor à esquerda é menor ou igual ao valor à direita.

Exemplo: 10 <= 10 é verdadeiro.

Prática 3

Faça comparações com as suas variáveis criadas até agora e observe o resultado no console do seu navegador.

Operadores lógicos

“AND” lógico (&&):

Retorna verdadeiro (true) se ambos os operandos forem verdadeiros.

“OR” lógico (||):

Retorna verdadeiro (true) se pelo menos um dos operandos for verdadeiro.

“NOT” lógico (!):

Inverte o valor de um operando booleano, tornando verdadeiro em falso e vice-versa.

Conversão de dados

1. parseInt():

- Converte uma string em um número inteiro.

- Exemplo: `parseInt("42")` retorna `42`.

2. parseFloat():

-Converte uma string em um número decimal.

- Exemplo: `parseFloat("3.14")` retorna `3.14`.

3. Number():

- Este construtor de objeto converte um valor em um número.

- Exemplo: `Number("123")` retorna `123`.

4. String():

- Este construtor de objeto converte um valor em uma string.

- Exemplo: `String(42)` retorna `"42"`.

5. Boolean():

- Este construtor de objeto converte um valor em um booleano (true ou false).

- Exemplo: `Boolean(0)` retorna `false`.

Conversão de dados

6. toString():

- Converte um valor em uma representação de string.
- Exemplo: `(42).toString()` retorna `"42"`.

7. toFixed():

- Este método converte um número em uma string com um número fixo de casas decimais.
- Exemplo: `(3.14159).toFixed(2)` retorna `"3.14"`.

8. toPrecision():

- Este método converte um número em uma string com um comprimento total específico (incluindo casas decimais).
- Exemplo: `(42.12345).toPrecision(5)` retorna `"42.123"`.

9. JSON.stringify():

- Este método converte um objeto JavaScript em uma string JSON.
- Exemplo: `JSON.stringify({ name: "John", age: 30 })` retorna `'{"name":"John","age":30}'`.

10. JSON.parse():

- Este método converte uma string JSON em um objeto JavaScript.
- Exemplo: `JSON.parse('{"name":"Jane","age":25}')` retorna um objeto com as propriedades `name` e `age`.

Prática 4

Declare uma variável “number” e a converta em texto

Declare uma variável “string” e a converta em number

Mostre-as no console usando console.log(typeof (SUA VARIÁVEL))

Estruturas de condição (Paramos aqui)

if/else

A estrutura if/else é a estrutura de condição mais básica. Ela permite executar um bloco de código se uma determinada condição for verdadeira, ou outro bloco de código se a condição for falsa

```
let idade = 18;


if (idade >= 18) {
  console.log("Você é maior de idade.");
} else {
  console.log("Você é menor de idade.");
}
```

O if/else if permite executar um bloco de código se uma determinada condição for verdadeira, ou outro bloco se a condição for falsa. Se a condição inicial for falsa, você pode verificar outras condições usando else if. Quando alguma condição for verdadeira, aquele bloco será executado e a cadeia de verificações será quebrada.

```
let nota = 5;
if (nota >= 9) {
  console.log("A nota é A.");
} else if (nota >= 7) {
  console.log("A nota é B.");
} else if (nota >= 5) {
  console.log("A nota é C.");
} else {
  console.log("A nota é D ou E.");
}
```

Estruturas de condição

A estrutura switch/case permite que o programador execute um bloco de código específico dependendo do valor de uma variável.

```
let dia = "segunda-feira";
switch (dia) {
  case "segunda-feira":
    console.log("Hoje é segunda-feira.");
    break;
  case "terça-feira":
    console.log("Hoje é terça-feira.");
    break;
  case "quarta-feira":
    console.log("Hoje é quarta-feira.");
    break;
  default:
    console.log("Hoje é um dia da semana.");
    break;
} //Atenção ao “break” em cada bloco
```

Condições ternárias

As condições ternárias são uma forma concisa de escrever uma estrutura if/else.

```
let idade = 18;
let resultado = idade >= 18 ? "maior de idade" : "menor de idade";

console.log(resultado); // Resulta em "maior de idade"
```

Prática 5

Crie uma condição para determinar se o numero é par ou impar e exiba a resposta em um alert.

Dica: use “alert(paramentros)”

Laços de repetição

while

Executa um bloco de código repetidamente, enquanto uma determinada condição for verdadeira.

```
let i = 0;
while (i < 10) {
  console.log(i);
  i++;
}
```

do...while

A estrutura do...while é semelhante à estrutura while, mas o bloco de código é executado pelo menos uma vez, antes que a condição seja verificada.

```
let i = 0;
do {
  console.log(i);
  i++;
} while (i < 10);
```

Laços de repetição

for

É uma estrutura de repetição versátil que permite especificar o início, o fim e o incremento do loop. Geralmente é utilizado, quando precisamos saber qual é o index atual da repetição.

```
for (let i = 0; i < 10; i++) {
  console.log(i);
}
```

Prática 6

Crie um laço de repetição para contar de 0 a “X” exibindo a contagem no console.

X = qualquer valor number

Desafio: quando a contagem acabar, informe em um alert.

Funções

Funções são blocos de código que podem ser reutilizados, com elas podemos organizar melhor o código, melhorar a legibilidade e a manutenibilidade do código, e reduzir a duplicação de código.

Funções são declaradas usando a palavra-chave “function”.

```
function nomeDaFuncao(argumentos) {
  // Corpo da função
}
```

Observe o nome da função, por boas práticas, devemos declarar a função com a primeira letra minuscula e as palavras seguintes com a primeira letra maiúscula. Os nomes não podem possuir caráteres especiais, não podem começar por números e devem condizer com o objetivo da função.

Funções

Argumentos são valores que são passados para uma função quando ela é chamada. A lista de argumentos é especificada entre os parênteses após o nome da função.

```
function soma(a, b) {
  return a + b;
}

console.log(soma(10, 20)); // 30
```

Funções

Corpo da função

O corpo da função é o bloco de código que é executado quando a função é chamada. O corpo da função é especificado entre chaves { }.

Retorno

Funções podem retornar um valor. O valor de retorno é especificado usando a instrução return.

```
function soma(a, b) {
  return a + b;
}
```

```
let resultado = soma(10, 20);
```

console.log(resultado); retorno no console seria 30

Funções

Funções de flecha são um tipo de função de expressão que foi introduzido no ECMAScript 6. Elas são uma forma concisa de escrever funções que aceitam um único argumento.

```
const soma = (a, b) => a + b;

console.log(soma(10, 20));
```

console mostraria 30

Prática 7

Crie uma função para determinar o fatorial de um número recebido por parâmetro e mostre o resultado.

Tipos de variáveis

Object: variável que contem uma coleção de atributos, para um mesmo objeto.

- Ex: const pessoa = {

  name: “nome”,

  age: 999

  }

Observe a virgula no final de cada atributo declarado. Para acessar o dado desse tipo de variável você deve usar a seguinte sintaxe: pessoa.name. Por exemplo: console.log(pessoa.name) o retorno seria “nome”.

Tipos de variáveis

E por ultimo temos o array, que é uma “coleção de valores”.

Por exemplo:

const frutas = [“banana”, “melão”, “laranja”]

Para acessar os dados você deve usar o index para a informação desejada, por exemplo, para receber o valor “melão” você deve fazer dessa forma “**frutas[1]**”.

Em um array os itens são contatos a partir do 0.

Você pode criar até mesmo um objeto de arrays...

Por exemplo: let objeto = {

frutas : [“banana”, “melão”, “laranja”],

vegetal : [“alface”, “cenoura”],

}

E para acessar os dados usaríamos: console.log(**objeto.frutas[0]**)

Tipos de variáveis

Ou o um arrays de objetos...

Por exemplo: let arrayDeObjetos = [

```
    { nome: 'João', idade: 25 },
    { nome: 'Maria', idade: 30 },
    { nome: 'Pedro', idade: 22 }
];
```

E para acessar os dados usaríamos:

```
console.log(arrayDeObjetos[0].nome);
```

Prática 8

Declare uma variável de cada tipo abaixo.

- Object
- Array

Agora, com o “console.log” acesse a primeira propriedade do seu object e o segundo item do seu array.

Métodos de arrays

Para os exemplos considere: let arr = [1, 2, 3, 4, 5]

push(): adiciona um elemento ao final do array.

Exemplo:

```
arr.push(6);
```

console.log(arr); resultado seria [1, 2, 3, 4, 5, 6]

pop(): remove o último elemento do array.

Exemplo:

```
arr.pop();
```

console.log(arr); resultado seria [1, 2, 3, 4, 5] – Removemos o 6, que seria o ultimo elemento.

Métodos de arrays

Para os exemplos considere: let arr = [1, 2, 3, 4, 5]

unshift(): adiciona um elemento ao início do array.

Exemplo:

```
arr.unshift(0);
```

console.log(arr); resultado= [0, 1, 2, 3, 4, 5]

shift(): remove o primeiro elemento do array.

Exemplo:

```
arr.shift();
```

console.log(arr); resultado = [1, 2, 3, 4, 5] – Removemos o 0 que adicionamos anteriormente

Métodos de arrays

Para os exemplos considere: let arr = [1, 2, 3, 3, 4, 5]

length: retorna o tamanho do array.

Exemplo:

console.log(arr.length); resposta no console “6”

indexOf(): retorna o índice do primeiro elemento que corresponde a um valor especificado.

Exemplo:

console.log(arr.indexOf(3)); resposta no console “2”

lastIndexOf(): retorna o índice do último elemento que corresponde a um valor especificado.

Exemplo:

console.log(arr.lastIndexOf(3)); resposta no console “3”

Métodos de arrays

Para os exemplos considere: let arr = [1, 2, 3, 4, 5]

map() : aplica uma função a cada elemento de um array e retorna um novo array com os resultados.

Exemplo:

```
let newArr = arr.map(x => x * 2);

console.log(newArr); resposta do console: [2, 4, 6, 8, 10]
```

filter(): filtra um array, mantendo apenas os elementos que satisfazem uma condição. A condição passada é avaliada para cada elemento do array e, se a condição for verdadeira, o elemento é mantido no novo array.

Exemplo:

```
let newArr = arr.filter(x => x % 2 === 0);

console.log(newArr); resposta do console: [2, 4]
```

reduce(): combina os elementos de um array em um único valor. Recebe dois argumentos: o valor acumulado e o elemento atual do array. O valor acumulado é inicializado com um valor inicial, nesse caso 0.

Exemplo:.

```
let sum = arr.reduce((acc, x) => acc + x, 0);

console.log(sum); retorno no console 15
```

Prática 9

Verifique o tamanho do seu array declarado no exercício 8.

Agora adicione mais um elemento a ele.

Agora busque pelo index de um valor especifico.

Funções assíncronas (Paramos aqui)

Funções assíncronas são funções que podem executar tarefas que não são bloqueantes. Isso significa que a execução da função não impede que outras tarefas sejam executadas. Elas são necessárias para lidar com tarefas que podem levar algum tempo para serem concluídas, como fazer uma solicitação HTTP ou ler um arquivo. Se essas tarefas fossem executadas de forma síncrona, elas bloqueariam a execução do código até que fossem concluídas.

Parar criar funções assíncronas podemos usar um desses dois esses dois métodos:

Usando a palavra-chave async: A palavra-chave async indica que a função é assíncrona.

Usando um objeto Promise: Um “Promise” é um tipo de dados que representa um resultado futuro.

Funções assíncronas

Funções async são funções que retornam um objeto Promise. O objeto é resolvido quando a tarefa assíncrona é concluída. Por exemplo:

```
async function getUserData() {
  // Faz uma solicitação HTTP para obter os dados do usuário
  fetch("https://api.example.com/users/123")
.then(response => {
response.json()
        dados = response
        console.log("Dados vindos de dentro da função: " + dados)
}) // Quando a resposta chegar, converte-a em JSON e grava as informações em “dados”
}).catch(error => console.log(error)) //Exibe erros no console, caso ocorram...
```

Funções assíncronas

Objetos “Promise” representam um resultado futuro. Eles podem ser usados para lidar com tarefas assíncronas.

```
const promise = new Promise((resolve, reject) => {
  // Faz uma solicitação HTTP para obter os dados do usuário
  fetch("https://api.example.com/users/123")
    .then((response) => resolve(response.json()))
    .catch((error) => reject(error));
});
// Resolve a promise quando a tarefa assíncrona é concluída
promise.then((data) => console.log(data));
```

Prática 10

Neste exercício vamos usar uma API “fake” para fazer a nossa requisição.

https://jsonplaceholder.typicode.com/users

Faça uma requisição para esse endereço passando o id desejado (0 a 10) recebido por parametro, após receber resposta exiba as informações no console.

Dica: para concatenar variável e texto você pode usar ` e ${sua variável}. Por exemplo: `https://jsonplaceholder.typicode.com/users/${id}`

Desafio: exiba apenas o nome e o endereço (rua) em um alert, com uma mensagem personalizada.

Dica: observe o tipo de resposta obtido para concluir o desafio...

Manipulação de elementos DOM

Para aprendermos sobre manipulação de DOM, primeiro vamos criar outro arquivo .js, para atrasarmos a sua execução.

Vamos fazer isso para facilitar e evitar erros, isso porque o nosso JavaScript está sendo executado ANTES do carregamento completo do nosso HTML.

Manipulação de elementos DOM

O DOM (Document Object Model) é uma representação orientada a objetos de uma página da web. Ele permite que os desenvolvedores manipulem os elementos da página, como texto, imagens, formulários e links.

Os elementos DOM são representados por objetos. Cada objeto DOM tem propriedades e métodos que podem ser usados para manipular o elemento.

Exemplos:

```
// Define “element” como um “ponteiro” ao PRIMEIRO elemento referenciado como parâmetro.

const element = document.querySelector("h1");


// Obtém o texto do elemento

const text = element.textContent;


// Define o texto do elemento

element.textContent = "Novo texto";
```

Manipulação de elementos DOM

Os métodos DOM são usados para executar ações no elemento.

Exemplos:

// Define “element” como um “ponteiro” ao PRIMEIRO elemento referenciado como parâmetro.

const element = document.querySelector("h1");

// Adiciona um novo filho ao elemento

element.appendChild(document.createTextNode("Novo filho"));

// Remove um filho do elemento

element.removeChild(element.firstChild);

Manipulação de elementos DOM

Os métodos DOM são usados para executar ações no elemento.

Exemplos de métodos para manipular **atributos**:

attributes: Retorna um objeto que representa os atributos do elemento.

setAttribute(): Define o valor de um atributo.

```
element.setAttribute("data-id", "123");
```

getAttribute(): Obtém o valor de um atributo. Nesse caso receberia 123

```
const id = element.getAttribute("data-id");
```

hasAttribute(): Verifica se um elemento possui um atributo específico.

```
const hasIdAttribute = element.hasAttribute("data-id");
```

removeAttribute(): Remove um atributo de um elemento.

```
element.removeAttribute("data-id");
```

Manipulação de elementos DOM

Os métodos DOM são usados para executar ações no elemento.

Exemplos de métodos para manipular CSS:

classList: Retorna um objeto que representa a lista de classes de um elemento.

classList.add(): Adiciona uma classe a um elemento.

```
element.classList.add("importante");
```

classList.remove(): Remove uma classe de um elemento.

```
element.classList.remove("importante");
```

classList.toggle(): Adiciona ou remove uma classe de um elemento, dependendo de sua presença atual.

```
element.classList.toggle("importante");
```

Manipulação de elementos DOM

A manipulação de elementos DOM pode ser feita usando uma variedade de métodos. Os mais comuns são:

querySelector(): Retorna o primeiro elemento que corresponde a um determinado seletor CSS.

querySelectorAll(): Retorna um array de elementos que correspondem a um determinado seletor CSS.

appendChild(): Adiciona um novo filho a um elemento.

removeChild(): Remove um filho de um elemento.

textContent: Obtém ou define o texto do elemento.

Manipulação de elementos DOM

Exemplo para adicionar um novo elemento à página:

```
// Define “element” como sendo uma div
const element = document.createElement("div");
//Adiciona texto ao novo elemento criado
element.textContent = "Novo elemento";
//Adiciona a nossa div criada ao body no HTML.
document.body.appendChild(element);
```

Manipulação de elementos DOM

Exemplo para remover um elemento da página:

```
//Define “element” como um “ponteiro” para remover o PRIMEIRO elemento com a classe “remover”

const element = document.querySelector(".remover");

//Remove o PRIMEIRO elemento que possui essa classe

element.parentNode.removeChild(element);
```

Manipulação de elementos DOM

Exemplo para alterar o texto de um elemento:

```
//Define “element” como um “ponteiro” para h1
const element = document.querySelector("h1");
//Atualiza o texto contido em h1, caso não haja, o texto será criado.
element.textContent = "Novo texto";
```

Manipulação de elementos DOM

Exemplo para adicionar um evento a um elemento:

```
//Define “element” como um “ponteiro” para “button”
const element = document.querySelector("button");


//Adiciona um listener ao botão.
element.addEventListener("click", () => {
  // Instruções a serem executadas
});
```

Prática 11

Adicione um texto a um “h1” (ou outro qualquer) em seu HTML.

O código deve ser feito em seu novo arquivo “.js”

Dica: Ao importar seu novo arquivo use “defer” após as declarações, isso fará com que o seu JS seja executado após a montagem da página OU chame seu novo arquivos “.js” logo após o elemento recém criado.

Desafio final

Com tudo aprendido até agora.

Crie um formulário para capturar um número (de 1 a 10)

Crie um botão para chamar uma função que faça uma requisição a nossa API usada anteriormente, passando o ID fornecido no campo.

Quando a resposta da requisição chegar, mostre em tela, o nome, e-mail e rua do usuário.

É recomendado criar um novo arquivo “.js” para esse exercicio

Referências

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Grammar_and_Types

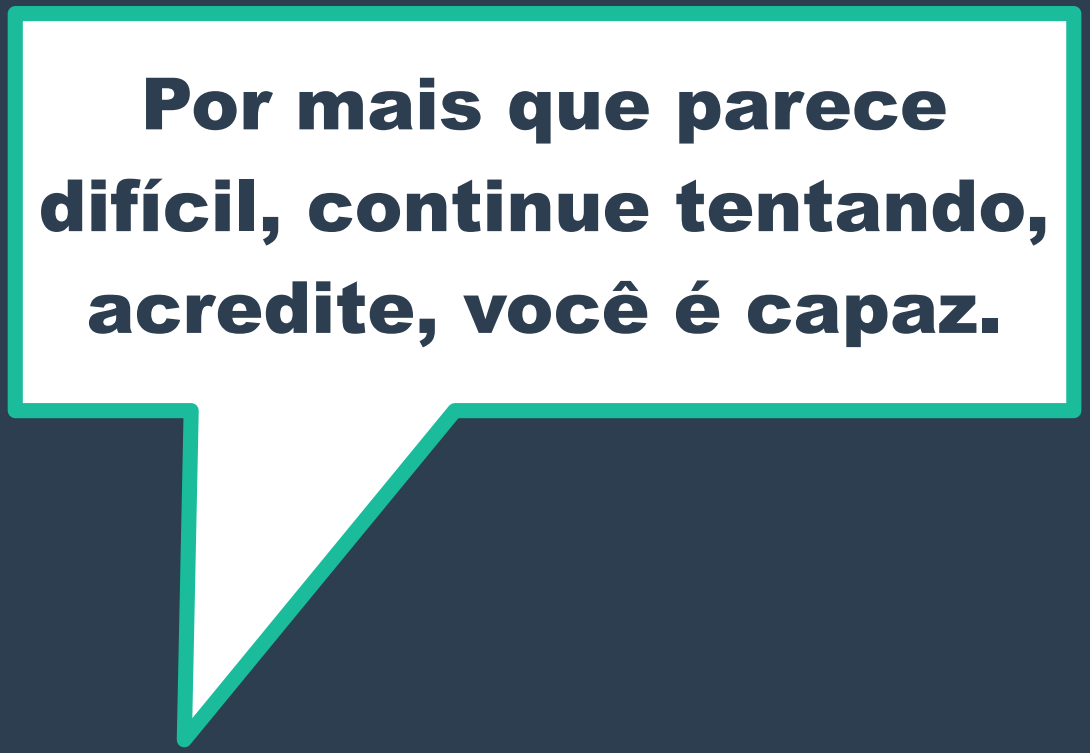
https://www.w3schools.com/js/

https://github.com/reprograma/On2-logica-com-js

https://youtu.be/OYPbr6ZG3pc?si=zh5EDEZobeD221se

https://youtu.be/WRlfwBof66s?si=7klZ6TGSU0j68XSS

https://roadmap.sh/javascript


Por mais que parece difícil, continue tentando, acredite, você é capaz.