**Exercícios de JavaScript - Do Básico ao Avançado (30 Questões)**

Olá! Este documento foi criado para te ajudar a praticar JavaScript, começando pelos fundamentos e avançando gradualmente para tópicos mais complexos. Cada seção apresentará um problema resolvido como exemplo e, em seguida, um problema semelhante para você resolver. Boa sorte!

**Nível Básico (10 Questões)**

**1. Variáveis e Tipos de Dados**

**Exemplo Resolvido: Declarando Variáveis e Atribuindo Valores**

**Problema:** Declare duas variáveis, uma para armazenar um nome (string) e outra para armazenar uma idade (número). Em seguida, exiba uma mensagem no console combinando essas variáveis.

*// Declaração da variável 'nome' e atribuição de um valor string* **let** nome = "Maria";

*// Declaração da variável 'idade' e atribuição de um valor numérico* **let** idade = 30;

*// Exibindo a mensagem no console usando template literals (crases)* console.log(`Olá, meu nome é $`{nome} e eu tenho `**${**idade**}** anos.`); *// Saída esperada: Olá, meu nome é Maria e eu tenho 30 anos.*

**Explicação:** Utilizamos let para declarar variáveis que podem ter seus valores alterados. nome recebeu uma string (texto entre aspas) e idade recebeu um number (número). O console.log é usado para exibir informações no console do navegador ou Node.js. As crases ( ` ) permitem criar strings com variáveis embutidas ( ${variavel} ).

**Seu Desafio: Calculando a Soma de Dois Números**

**Problema:** Declare duas variáveis, numero1 e numero2 , e atribua a elas valores numéricos de sua escolha. Crie uma terceira variável, soma , que armazene o resultado da adição de numero1 e numero2 . Por fim, exiba o valor de soma no console.

**2. Operadores Aritméticos**

**Exemplo Resolvido: Usando Operadores Básicos**

**Problema:** Dadas duas variáveis numéricas, realize as operações de adição, subtração, multiplicação e divisão, exibindo o resultado de cada uma.

**let** a = 10;

**let** b = 5;

console.log(`Adição: **${**a + b**}**`); *// Saída: Adição: 15*

console.log(`Subtração: **${**a - b**}**`); *// Saída: Subtração: 5*

console.log(`Multiplicação: **${**a \* b**}**`); *// Saída: Multiplicação: 50* console.log(`Divisão: **${**a / b**}**`); *// Saída: Divisão: 2*

console.log(`Resto da Divisão: **${**a % b**}**`); *// Saída: Resto da Divisão: 0*

**Explicação:** JavaScript suporta os operadores aritméticos básicos: + (adição), - (subtração), \* (multiplicação), / (divisão) e % (resto da divisão). Eles funcionam como na matemática.

**Seu Desafio: Calculando a Média de Três Notas**

**Problema:** Declare três variáveis, nota1 , nota2 e nota3 , e atribua a elas valores numéricos que representem notas de um aluno. Calcule a média dessas três notas e armazene o resultado em uma variável media . Exiba a media no console.

**3. Estruturas Condicionais (if/else)**

**Exemplo Resolvido: Verificando se um Número é Par ou Ímpar**

**Problema:** Dada uma variável numero , verifique se ela é par ou ímpar e exiba a mensagem correspondente no console.

**let** numero = 7;

**if** (numero % 2 === 0) {

console.log(`**${**numero**}** é um número par.`);

} **else** {

console.log(`**${**numero**}** é um número ímpar.`);

}

*// Saída esperada: 7 é um número ímpar.*

numero = 10;

**if** (numero % 2 === 0) {

console.log(`**${**numero**}** é um número par.`);

} **else** {

console.log(`**${**numero**}** é um número ímpar.`);

}

*// Saída esperada: 10 é um número par.*

**Explicação:** O operador % (resto da divisão) é fundamental aqui. Se o resto da divisão por 2 for 0, o número é par. Caso contrário, é ímpar. A estrutura if/else permite executar blocos de código diferentes com base em uma condição ( numero % 2 === 0 ). O === é o operador de igualdade estrita, que compara valor e tipo.

**Seu Desafio: Verificando a Idade para Votar**

**Problema:** Declare uma variável idade e atribua a ela um valor numérico. Usando uma estrutura if/else , verifique se a pessoa tem idade suficiente para votar (considerando 16 anos como idade mínima). Exiba uma mensagem no console informando se a pessoa pode ou não votar.

**4. Loops (for)**

**Exemplo Resolvido: Contando de 1 a 5**

**Problema:** Use um loop for para exibir os números de 1 a 5 no console.

**for** (**let** i = 1; i <= 5; i++) {

console.log(i);

}

*/\**

*Saída esperada:*

*1*

*2*

*3*

*4*

*5*

*\*/*

**Explicação:** O loop for é ideal para repetir um bloco de código um número conhecido de vezes. Ele tem três partes: inicialização ( let i = 1 ), condição ( i <= 5 ) e incremento ( i++ ). O código dentro das chaves é executado enquanto a condição for verdadeira.

**Seu Desafio: Tabuada do 7**

**Problema:** Use um loop for para gerar e exibir a tabuada do número 7 (de 7x1 até 7x10) no console. Cada linha deve ter o formato 7 x N = Resultado .

**5. Loops (while)**

**Exemplo Resolvido: Contagem Regressiva**

**Problema:** Use um loop while para fazer uma contagem regressiva de 5 até 1.

**let** contador = 5;

**while** (contador >= 1) {

console.log(contador);

contador--; *// Decrementa o contador a cada iteração*

}

*/\**

*Saída esperada:*

*5*

*4*

*3*

*2*

*1*

*\*/*

**Explicação:** O loop while continua executando um bloco de código enquanto uma condição especificada for verdadeira. É crucial garantir que a condição eventualmente se torne falsa para evitar loops infinitos. Neste caso, contador-- garante que contador >= 1 se torne falso em algum momento.

**Seu Desafio: Acumulando Valores até um Limite**

**Problema:** Crie uma variável total inicializada com 0 e uma variável limite com o valor 20. Usando um loop while , adicione números inteiros sequenciais (começando do 1) a total até que total seja maior ou igual a limite . A cada adição, exiba o valor atual de total no console.

**6. Funções Básicas**

**Exemplo Resolvido: Saudação Simples**

**Problema:** Crie uma função que receba um nome como parâmetro e retorne uma saudação personalizada.

**function** saudar(nome) {

**return** `Olá, **${**nome**}**! Bem-vindo(a)!`;

}

*// Chamando a função e exibindo o resultado*

console.log(saudar("Ana")); *// Saída: Olá, Ana! Bem-vindo(a)!*

console.log(saudar("Pedro")); *// Saída: Olá, Pedro! Bem-vindo(a)!*

**Explicação:** Funções são blocos de código reutilizáveis. Elas podem receber parâmetros (como nome ) e retornar um valor. A palavra-chave function define a função, seguida pelo nome da função e os parâmetros entre parênteses. O return especifica o valor que a função deve produzir.

**Seu Desafio: Calculadora de Quadrado**

**Problema:** Crie uma função chamada calcularQuadrado que receba um número como parâmetro e retorne o quadrado desse número (o número multiplicado por ele mesmo). Teste a função com alguns valores e exiba os resultados no console.

**7. Arrays (Introdução)**

**Exemplo Resolvido: Acessando Elementos de um Array**

**Problema:** Dado um array de frutas, exiba a primeira e a última fruta no console.

**const** frutas = ["Maçã", "Banana", "Laranja", "Uva"];

*// Acessando o primeiro elemento (índice 0)*

console.log(`Primeira fruta: **${**frutas[0]**}**`); *// Saída: Primeira fruta: Maçã*

*// Acessando o último elemento (usando .length - 1)*

console.log(`Última fruta: **${**frutas[frutas.length - 1]**}**`); *// Saída: Última fruta: Uva*

**Explicação:** Arrays são listas ordenadas de valores. Cada valor tem um índice numérico, começando do 0. array.length retorna o número de elementos no array, e

array.length - 1 nos dá o índice do último elemento.

**Seu Desafio: Listando Itens de Compras**

**Problema:** Crie um array chamado listaDeCompras com pelo menos 4 itens (strings). Usando um loop for , percorra o array e exiba cada item no console, um por linha.

**8. Objetos (Introdução)**

**Exemplo Resolvido: Acessando Propriedades de um Objeto**

**Problema:** Dado um objeto que representa uma pessoa, exiba o nome e a idade dessa pessoa.

**const** pessoa = {

nome: "Carlos",

idade: 28,

cidade: "São Paulo"

};

*// Acessando propriedades usando a notação de ponto*

console.log(`Nome: **${**pessoa.nome**}**`); *// Saída: Nome: Carlos*

console.log(`Idade: **${**pessoa.idade**}**`); *// Saída: Idade: 28*

*// Também é possível acessar usando a notação de colchetes (útil para nomes de propriedades com espaços ou variáveis)*

console.log(`Cidade: **${**pessoa["cidade"]**}**`); *// Saída: Cidade: São Paulo*

**Explicação:** Objetos são coleções de pares chave-valor. As chaves (ou propriedades) são strings (ou símbolos) e os valores podem ser de qualquer tipo de dado. Podemos acessar os valores usando a notação de ponto ( objeto.propriedade ) ou a notação de colchetes ( objeto["propriedade"] ).

**Seu Desafio: Informações de um Livro**

**Problema:** Crie um objeto chamado livro com as seguintes propriedades: titulo (string), autor (string) e anoPublicacao (número). Atribua valores a essas propriedades. Em seguida, exiba no console uma frase que combine o título e o autor do livro.

**9. Operadores Lógicos**

**Exemplo Resolvido: Verificando Condições Múltiplas**

**Problema:** Verifique se um usuário é maior de idade (18 anos ou mais) E tem uma carteira de motorista (booleano).

**let** idadeUsuario = 20;

**let** temCarteira = **true**;

**if** (idadeUsuario >= 18 && temCarteira) {

console.log("Usuário pode dirigir.");

} **else** {

console.log("Usuário NÃO pode dirigir.");

}

*// Saída esperada: Usuário pode dirigir.*

idadeUsuario = 17;

temCarteira = **true**;

**if** (idadeUsuario >= 18 && temCarteira) {

console.log("Usuário pode dirigir.");

} **else** {

console.log("Usuário NÃO pode dirigir.");

}

*// Saída esperada: Usuário NÃO pode dirigir.*

**Explicação:** Os operadores lógicos && (AND), || (OR) e ! (NOT) são usados para combinar ou negar condições. && retorna true se AMBAS as condições forem verdadeiras. || retorna true se PELO MENOS UMA das condições for verdadeira. ! inverte o valor booleano.

**Seu Desafio: Verificando Acesso ao Sistema**

**Problema:** Crie duas variáveis: usuarioLogado (booleano) e isAdmin (booleano). Usando operadores lógicos, verifique se o usuário está logado OU é um administrador. Exiba uma mensagem no console informando se o usuário tem acesso total ao sistema.

**10. Switch Case**

**Exemplo Resolvido: Classificando Dias da Semana**

**Problema:** Dada uma variável diaDaSemana (número de 1 a 7), use switch case para exibir o nome do dia correspondente.

**let** diaDaSemana = 3;

**let** nomeDoDia;

**switch** (diaDaSemana) {

**case** 1:

nomeDoDia = "Domingo";

**break**;

**case** 2:

nomeDoDia = "Segunda-feira";

**break**;

**case** 3:

nomeDoDia = "Terça-feira";

**break**;

**case** 4:

nomeDoDia = "Quarta-feira";

**break**;

**case** 5:

nomeDoDia = "Quinta-feira";

**break**;

**case** 6:

nomeDoDia = "Sexta-feira";

**break**;

**case** 7:

nomeDoDia = "Sábado";

**break**;

**default**:

nomeDoDia = "Dia inválido";

}

console.log(`Hoje é: **${**nomeDoDia**}**`); *// Saída esperada: Hoje é: Terça-feira*

**Explicação:** A estrutura switch case é uma alternativa ao if/else if quando você tem muitas condições baseadas no valor de uma única variável. O break é crucial para sair do switch após encontrar uma correspondência. O default é executado se nenhuma correspondência for encontrada.

**Seu Desafio: Classificando Notas**

**Problema:** Dada uma variável nota (número inteiro de 0 a 100), use switch case para classificar a nota em conceitos: "A" (90-100), "B" (80-89), "C" (70-79), "D" (60-69) e "F" (abaixo de 60). Exiba o conceito no console. (Dica: você pode agrupar case s ou usar if/else dentro de case s para ranges, mas tente pensar em como o switch pode ser usado de forma mais direta para ranges, talvez com uma pequena adaptação na variável de controle).

**Nível Intermediário (10 Questões)**

**11. Funções (Parâmetros e Retorno)**

**Exemplo Resolvido: Calculadora de IMC**

**Problema:** Crie uma função que calcule o Índice de Massa Corporal (IMC) de uma pessoa, recebendo peso (em kg) e altura (em metros) como parâmetros, e retorne o valor do IMC.

**function** calcularIMC(peso, altura) {

*// IMC = peso / (altura \* altura)*

**const** imc = peso / (altura \* altura);

**return** imc;

}

*// Testando a função*

**let** pesoPessoa = 70; *// kg*

**let** alturaPessoa = 1.75; *// metros*

**let** imcPessoa = calcularIMC(pesoPessoa, alturaPessoa);

console.log(`O IMC para uma pessoa com $`{pesoPessoa}kg e `**${**alturaPessoa**}**m é: **${**imcPessoa.toFixed(2)**}**`);

*// Saída esperada: O IMC para uma pessoa com 70kg e 1.75m é: 22.86*

**Explicação:** Esta função demonstra como funções podem receber múltiplos parâmetros e realizar cálculos mais complexos antes de retornar um resultado. O método .toFixed(2) é usado para formatar o número com duas casas decimais.

**Seu Desafio: Conversor de Temperatura**

**Problema:** Crie uma função chamada converterCelsiusParaFahrenheit que receba uma temperatura em Celsius como parâmetro e retorne o valor equivalente em Fahrenheit. A fórmula de conversão é: Fahrenheit = Celsius \* 9/5 + 32 . Teste a função com alguns valores e exiba os resultados no console.

**12. Arrays (Métodos Essenciais: push, pop, shift, unshift)**

**Exemplo Resolvido: Adicionando e Removendo Elementos**

**Problema:** Dado um array de tarefas, adicione uma nova tarefa ao final e remova a primeira tarefa.

**let** tarefas = ["Estudar JavaScript", "Fazer compras", "Pagar contas"];

console.log("Tarefas iniciais:", tarefas);

*// Saída: Tarefas iniciais: [ 'Estudar JavaScript', 'Fazer compras', 'Pagar contas' ]*

*// Adicionar uma tarefa ao final do array com push()*

tarefas.push("Lavar o carro");

console.log("Após adicionar 'Lavar o carro':", tarefas);

*// Saída: Após adicionar 'Lavar o carro': [ 'Estudar JavaScript', 'Fazer compras', 'Pagar contas', 'Lavar o carro' ]*

*// Remover a primeira tarefa do array com shift()*

tarefas.shift();

console.log("Após remover a primeira tarefa:", tarefas);

*// Saída: Após remover a primeira tarefa: [ 'Fazer compras', 'Pagar contas', 'Lavar o carro' ]*

**Explicação:** push() adiciona um ou mais elementos ao final de um array e retorna o novo comprimento do array. shift() remove o primeiro elemento de um array e retorna esse elemento removido, e os elementos restantes são deslocados para uma posição inferior.

**Seu Desafio: Gerenciando uma Lista de Desejos**

**Problema:** Crie um array chamado listaDeDesejos com 3 itens. Adicione um novo item ao final, remova o último item e, por fim, adicione um item no início da lista. Exiba a lista após cada operação.

**13. Objetos (Manipulação e Iteração)**

**Exemplo Resolvido: Atualizando Propriedades de um Objeto**

**Problema:** Dado um objeto de usuário, atualize sua idade e adicione uma nova propriedade de email.

**let** usuario = {

nome: "João",

idade: 25,

cidade: "Rio de Janeiro"

};

console.log("Usuário inicial:", usuario);

*// Saída: Usuário inicial: { nome: 'João', idade: 25, cidade: 'Rio de Janeiro' }*

*// Atualizando a idade*

usuario.idade = 26;

*// Adicionando uma nova propriedade*

usuario.email = "joao@example.com";

console.log("Usuário atualizado:", usuario);

*// Saída: Usuário atualizado: { nome: 'João', idade: 26, cidade: 'Rio de Janeiro', email: 'joao@example.com' }*

**Explicação:** Propriedades de objetos podem ser facilmente atualizadas ou adicionadas usando a notação de ponto ou colchetes. Se a propriedade já existe, seu valor é sobrescrito; se não existe, ela é criada.

**Seu Desafio: Perfil de Rede Social**

**Problema:** Crie um objeto perfil com as propriedades nome , seguidores (número) e seguindo (número). Atribua valores iniciais. Em seguida, aumente o número de seguidores em 100 e adicione uma nova propriedade bio (string) com uma breve descrição. Exiba o objeto perfil completo após as modificações.

**14. Manipulação de Strings (Métodos: toLowerCase, toUpperCase, split, join, charAt, slice)**

**Exemplo Resolvido: Formatando Nomes**

**Problema:** Dada uma string com um nome completo, transforme-a para que todas as letras sejam minúsculas e, em seguida, apenas a primeira letra de cada palavra seja maiúscula.

**let** nomeCompleto = "aNa cArOlInA sIlVa";

*// 1. Converter para minúsculas*

**let** nomeMinusculo = nomeCompleto.toLowerCase();

console.log("Minúsculo:", nomeMinusculo);

*// Saída: Minúsculo: ana carolina silva*

*// 2. Capitalizar a primeira letra de cada palavra*

**let** nomeFormatado = nomeMinusculo.split(' ').map(palavra => {

**return** palavra.charAt(0).toUpperCase() + palavra.slice(1);

}).join(' ');

console.log("Formatado:", nomeFormatado);

*// Saída: Formatado: Ana Carolina Silva*

**Explicação:** toLowerCase() converte a string para minúsculas. split(' ') divide a string em um array de palavras. map() é usado para aplicar uma função a cada palavra: charAt(0).toUpperCase() pega a primeira letra e a torna maiúscula, e

slice(1) pega o restante da palavra. join(' ') junta as palavras de volta em uma string.

**Seu Desafio: Invertendo uma String**

**Problema:** Crie uma função chamada inverterString que receba uma string como parâmetro e retorne a string invertida. Por exemplo, se a entrada for "javascript", a saída deve ser "tpircsavaj".

**15. Manipulação de Datas (Objeto Date)**

**Exemplo Resolvido: Obtendo a Data Atual**

**Problema:** Exiba a data e hora atuais no console.

**const** dataAtual = **new** Date();

console.log("Data e Hora Atuais:", dataAtual);

*// Saída esperada: Data e Hora Atuais: [data e hora atuais, ex: Sat Sep 21 2025 10:30:00 GMT-0300 (Brasilia Standard Time)]*

*// Para formatar de forma mais legível*

console.log("Data formatada:", dataAtual.toLocaleDateString('pt-BR')); console.log("Hora formatada:", dataAtual.toLocaleTimeString('pt-BR'));

**Explicação:** O objeto Date é usado para trabalhar com datas e horas. new Date() cria um objeto Date com a data e hora atuais. Métodos como toLocaleDateString() e toLocaleTimeString() ajudam a formatar a saída de acordo com a localidade.

**Seu Desafio: Calculando a Idade em Anos**

**Problema:** Crie uma função chamada calcularIdade que receba uma data de nascimento (como uma string no formato "AAAA-MM-DD") e retorne a idade da pessoa em anos completos. Considere a data atual como referência. Teste a função com algumas datas de nascimento.

**16. Métodos de Array (map, filter, reduce)**

**Exemplo Resolvido: Filtrando e Mapeando um Array de Objetos**

**Problema:** Dada uma lista de usuários (objetos com nome e idade ), filtre apenas os usuários maiores de 18 anos e retorne um novo array com os nomes desses usuários em maiúsculas.

**const** usuarios = [

{ nome: "Ana", idade: 22 },

{ nome: "Bruno", idade: 17 },

{ nome: "Carla", idade: 30 },

{ nome: "Daniel", idade: 19 }

];

**const** nomesMaioresDeIdade = usuarios

.filter(usuario => usuario.idade >= 18) *// Filtra usuários maiores de 18* .map(usuario => usuario.nome.toUpperCase()); *// Mapeia para nomes em maiúsculas*

console.log(nomesMaioresDeIdade);

*// Saída esperada: [ 'ANA', 'CARLA', 'DANIEL' ]*

**Explicação:** filter() cria um novo array com todos os elementos que passam em um teste. map() cria um novo array chamando uma função para cada elemento do array. A combinação desses métodos é muito comum para transformar dados.

**Seu Desafio: Calculando o Total de um Carrinho de Compras**

**Problema:** Dada uma lista de itens de um carrinho de compras (array de objetos, cada um com nome , preco e quantidade ), use o método reduce() para calcular o valor total do carrinho. Cada item deve ter seu preco multiplicado pela quantidade antes de ser somado ao total.

**17. Desestruturação de Objetos e Arrays**

**Exemplo Resolvido: Extraindo Propriedades de um Objeto**

**Problema:** Dado um objeto de configuração, extraia as propriedades host e port para variáveis separadas.

**const** config = {

host: "localhost",

port: 8080,

timeout: 5000

};

*// Desestruturação de objeto*

**const** { host, port } = config;

console.log(`Servidor: $`{host}:`**${**port**}**`);

*// Saída esperada: Servidor: localhost:8080*

**Explicação:** A desestruturação permite extrair valores de arrays ou propriedades de objetos em variáveis distintas de forma concisa. É uma sintaxe muito útil para tornar o código mais limpo e legível.

**Seu Desafio: Trocando Valores de Variáveis**

**Problema:** Dadas duas variáveis a e b , troque seus valores usando desestruturação de array, sem a necessidade de uma variável temporária.

**18. Spread Operator (...)**

**Exemplo Resolvido: Combinando Arrays**

**Problema:** Combine dois arrays em um único novo array.

**const** array1 = [1, 2, 3];

**const** array2 = [4, 5, 6];

**const** arrayCombinado = [...array1, ...array2];

console.log(arrayCombinado);

*// Saída esperada: [ 1, 2, 3, 4, 5, 6 ]*

**Explicação:** O spread operator ( ... ) permite expandir um iterável (como um array ou string) em locais onde zero ou mais argumentos (para chamadas de função) ou

elementos (para literais de array) são esperados. Ele também pode ser usado para copiar objetos e arrays.

**Seu Desafio: Adicionando Propriedades a um Objeto Existente**

**Problema:** Dado um objeto produto com nome e preco , crie um **novo** objeto produtoComEstoque que inclua todas as propriedades de produto e adicione uma nova propriedade estoque com um valor numérico. Garanta que o objeto original produto não seja modificado.

**19. Classes e Programação Orientada a Objetos (POO)**

**Exemplo Resolvido: Criando uma Classe Básica**

**Problema:** Crie uma classe Animal com um construtor que receba nome e especie , e um método fazerBarulho que exiba uma mensagem genérica.

**class** Animal {

**constructor**(nome, especie) {

**this**.nome = nome;

**this**.especie = especie;

}

fazerBarulho() {

console.log(`**${this**.nome**}** faz um barulho.`);

}

}

**const** meuCachorro = **new** Animal("Rex", "Cachorro");

meuCachorro.fazerBarulho(); *// Saída: Rex faz um barulho.*

**const** meuGato = **new** Animal("Miau", "Gato");

console.log(`$`{meuGato.nome} é um `**${**meuGato.especie**}**.`); *// Saída: Miau é um Gato.*

**Explicação:** Classes são "moldes" para criar objetos. O constructor é um método especial que é chamado quando um novo objeto é criado ( new Animal() ). this refere-se à instância atual do objeto. Métodos são funções associadas à classe.

**Seu Desafio: Classe Carro**

**Problema:** Crie uma classe Carro com um construtor que receba marca , modelo e ano . Adicione um método obterDetalhes que retorne uma string com todos os

detalhes do carro (ex: "Marca: Ford, Modelo: Fiesta, Ano: 2020"). Crie uma instância da classe e exiba seus detalhes.

**20. Promises e Async/Await**

**Exemplo Resolvido: Simulando uma Operação Assíncrona com Promise**

**Problema:** Crie uma função que simule o carregamento de dados após um atraso de 2 segundos, retornando uma Promise que resolve com uma mensagem de sucesso.

**function** carregarDados() {

**return new** Promise((resolve, reject) => {

setTimeout(() => {

**const** sucesso = **true**; *// Simula sucesso ou falha*

**if** (sucesso) {

resolve("Dados carregados com sucesso!");

} **else** {

reject("Falha ao carregar dados.");

}

}, 2000); *// 2 segundos de atraso*

});

}

console.log("Iniciando carregamento...");

carregarDados()

.then(mensagem => console.log(mensagem)) *// Saída após 2s: Dados carregados com sucesso!*

.**catch**(erro => console.error(erro));

console.log("Requisição enviada.");

**Explicação:** Promises representam um valor que pode estar disponível agora, no futuro ou nunca. resolve é chamado quando a operação assíncrona é bem-sucedida, e reject quando falha. setTimeout simula um atraso. then() lida com o sucesso, catch() com a falha.

**Seu Desafio: Função Assíncrona com Async/Await**

**Problema:** Reescreva o exemplo anterior ( carregarDados ) usando async/await para tornar o código assíncrono mais fácil de ler e escrever. A função deve ainda simular um atraso de 2 segundos e retornar a mesma mensagem de sucesso.

**Nível Avançado (10 Questões)**

**21. Manipulação do DOM (Criação e Remoção de Elementos)**

**Exemplo Resolvido: Criando e Adicionando um Novo Elemento**

**Problema:** Crie um novo elemento div com o texto "Eu sou uma nova div!" e adicione-o ao final do <body> do documento HTML.

*// 1. Cria um novo elemento <div>*

**const** novaDiv = document.createElement("div");

*// 2. Adiciona texto ao novo elemento*

novaDiv.textContent = "Eu sou uma nova div!";

*// 3. Adiciona o novo elemento ao final do <body>*

document.body.appendChild(novaDiv);

**Explicação:** document.createElement() cria um novo nó de elemento. appendChild() adiciona um nó como o último filho de um nó pai especificado. Isso é fundamental para construir interfaces dinâmicas.

**Seu Desafio: Removendo um Elemento da Página**

**Problema:** Suponha que você tenha um parágrafo HTML com o id paragrafoParaRemover . Usando JavaScript, remova este parágrafo da página.

**22. Event Delegation**

**Exemplo Resolvido: Lidando com Cliques em uma Lista Dinâmica**

**Problema:** Dada uma lista <ul> com vários itens <li> , e novos itens podem ser adicionados dinamicamente. Ao clicar em qualquer <li> , exiba o texto do item clicado no console, usando delegação de eventos.

*<!-- Seu arquivo HTML -->*

<**ul** id="minhaListaEventos">

<**li**>Item 1</**li**>

<**li**>Item 2</**li**>

<**li**>Item 3</**li**>

</**ul**>

<**button** id="adicionarItem">Adicionar Novo Item</**button**>

**const** listaEventos = document.getElementById("minhaListaEventos"); **const** botaoAdicionar = document.getElementById("adicionarItem"); **let** contadorItens = 3;

*// Adiciona um event listener ao elemento PAI (<ul>)*

listaEventos.addEventListener("click", **function**(event) {

*// Verifica se o clique ocorreu em um <li>*

**if** (event.target.tagName === "LI") {

console.log("Item clicado:", event.target.textContent);

}

});

*// Adiciona um novo item à lista dinamicamente*

botaoAdicionar.addEventListener("click", **function**() {

contadorItens++;

**const** novoLi = document.createElement("li");

novoLi.textContent = `Item **${**contadorItens**}**`;

listaEventos.appendChild(novoLi);

});

**Explicação:** Em vez de adicionar um event listener a cada <li> individualmente (o que seria ineficiente para listas grandes ou dinâmicas), adicionamos um único listener ao elemento pai ( <ul> ). Quando um clique ocorre, o evento "borbulha" até o pai, e podemos verificar event.target para saber qual elemento filho foi clicado.

**Seu Desafio: Delegação de Eventos para Botões de Excluir**

**Problema:** Crie uma lista <ul> onde cada <li> contém um texto e um botão "Excluir". Ao clicar em qualquer botão "Excluir", o item <li> correspondente deve ser removido da lista. Use delegação de eventos para lidar com os cliques nos botões.

**23. Formulários e Validação**

**Exemplo Resolvido: Validando um Campo de Email Simples**

**Problema:** Crie um formulário com um campo de email e um botão de submit. Ao tentar submeter o formulário, verifique se o campo de email não está vazio e se contém um @ .

*<!-- Seu arquivo HTML -->*

<**form** id="meuFormulario">

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

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

<**button** type="submit">Enviar</**button**>

<**p** id="mensagemErro" style="color: red;"></**p**>

</**form**>

**const** formulario = document.getElementById("meuFormulario");

**const** inputEmail = document.getElementById("email");

**const** mensagemErro = document.getElementById("mensagemErro");

formulario.addEventListener("submit", **function**(event) {

event.preventDefault(); *// Impede o envio padrão do formulário*

**const** email = inputEmail.value;

**if** (email === "") {

mensagemErro.textContent = "O campo de email não pode estar vazio."; } **else if** (!email.includes("@")) {

mensagemErro.textContent = "O email deve conter um @.";

} **else** {

mensagemErro.textContent = ""; *// Limpa a mensagem de erro*

alert("Formulário enviado com sucesso!");

*// Aqui você enviaria os dados para um servidor, por exemplo*

}

});

**Explicação:** event.preventDefault() é crucial para impedir que o formulário seja enviado e a página recarregada, permitindo que o JavaScript lide com a validação. A validação é feita verificando o value do input e usando métodos de string como includes() .

**Seu Desafio: Validando Senha e Confirmação de Senha**

**Problema:** Crie um formulário com dois campos de input: senha e confirmarSenha . Ao submeter o formulário, verifique se ambos os campos não estão vazios e se os valores digitados são idênticos. Exiba mensagens de erro apropriadas ou um alerta de sucesso.

**24. Requisições Assíncronas (Tratamento de Erros e Carregamento)**

**Exemplo Resolvido: Exibindo Status de Carregamento e Erros**

**Problema:** Faça uma requisição para uma API e exiba um status de "Carregando..." enquanto a requisição está pendente. Em caso de sucesso, exiba os dados; em caso de

erro, exiba uma mensagem de erro.

*<!-- Seu arquivo HTML -->*

<**button** id="carregarDadosBtn">Carregar Dados</**button**>

<**p** id="status"></**p**>

<**pre** id="resultado"></**pre**>

**const** carregarDadosBtn = document.getElementById("carregarDadosBtn"); **const** statusElement = document.getElementById("status");

**const** resultadoElement = document.getElementById("resultado");

carregarDadosBtn.addEventListener("click", **async** () => {

statusElement.textContent = "Carregando...";

resultadoElement.textContent = "";

**try** {

**const** response = **await**

fetch("https://jsonplaceholder.typicode.com/posts/1");

**if** (!response.ok) {

**throw new Error**(`Erro HTTP! Status: **${**response.status**}**`);

}

**const** data = **await** response.json();

statusElement.textContent = "Dados carregados com sucesso!";

resultadoElement.textContent = JSON.stringify(data, **null**, 2);

} **catch** (error) {

statusElement.textContent = "Erro ao carregar dados.";

resultadoElement.textContent = `Detalhes do erro: **${**error.message**}**`; console.error("Erro na requisição:", error);

}

});

**Explicação:** Usamos async/await para lidar com Promises de forma mais síncrona. O bloco try...catch é essencial para capturar e tratar erros que podem ocorrer durante a requisição. Atualizamos elementos do DOM para informar o usuário sobre o status da operação.

**Seu Desafio: Buscando Imagens Aleatórias**

**Problema:** Crie um botão e um elemento <img> no HTML. Ao clicar no botão, faça uma requisição para uma API de imagens aleatórias (ex: https://dog.ceo/api/breeds/image/random para imagens de cachorros). Exiba a imagem retornada no elemento <img> . Adicione um status de "Carregando imagem..." e trate possíveis erros, exibindo uma mensagem se a imagem não puder ser carregada.

**25. Módulos ES6 (Import/Export)**

**Exemplo Resolvido: Exportando e Importando Funções**

**Problema:** Crie dois arquivos JavaScript. Um ( matematica.js ) deve exportar funções de soma e subtração. O outro ( app.js ) deve importar e usar essas funções.

*// matematica.js*

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

**return** a + b;

}

**export function** subtrair(a, b) {

**return** a - b;

}

*// app.js*

**import** { somar, subtrair } **from** './matematica.js';

console.log("Soma:", somar(5, 3)); *// Saída: Soma: 8*

console.log("Subtração:", subtrair(10, 4)); *// Saída: Subtração: 6*

**Explicação:** Módulos ES6 permitem organizar o código em arquivos separados, tornando-o mais modular e reutilizável. export é usado para disponibilizar funções, variáveis ou classes de um módulo. import é usado para consumir esses itens em outro módulo. Para que isso funcione no navegador, o script app.js deve ser carregado com type="module" no HTML ( <script type="module" src="app.js"> </script> ).

**Seu Desafio: Módulo de Utilitários de String**

**Problema:** Crie um arquivo stringUtils.js que exporte duas funções: capitalizarPrimeiraLetra(string) (que torna a primeira letra de uma string maiúscula) e contarVogais(string) (que retorna o número de vogais em uma string). Em um arquivo main.js , importe e use essas funções para processar uma string de sua escolha.

**26. Manipulação de Erros (try...catch...finally)**

**Exemplo Resolvido: Capturando e Tratando Erros**

**Problema:** Crie uma função que divida dois números, mas que lance um erro se o divisor for zero. Use try...catch para lidar com esse erro.

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

**if** (b === 0) {

**throw new Error**("Divisão por zero não é permitida.");

}

**return** a / b;

}

**try** {

**let** resultado = dividir(10, 2);

console.log("Resultado da divisão:", resultado); *// Saída: Resultado da divisão: 5*

resultado = dividir(10, 0);

console.log("Este console.log não será executado.");

} **catch** (erro) {

console.error("Ocorreu um erro:", erro.message); *// Saída: Ocorreu um erro: Divisão por zero não é permitida.*

} **finally** {

console.log("Bloco finally sempre é executado.");

}

**Explicação:** O bloco try contém o código que pode gerar um erro. Se um erro ( throw new Error() ) ocorrer, a execução do try é interrompida e o controle é passado para o bloco catch , que recebe o objeto de erro. O bloco finally é opcional e sempre é executado, independentemente de ter ocorrido um erro ou não.

**Seu Desafio: Validando Entrada Numérica**

**Problema:** Crie uma função que receba uma string e tente convertê-la para um número. Se a conversão resultar em NaN (Not-a-Number), a função deve lançar um erro. Use try...catch para chamar essa função e lidar com o erro, exibindo uma mensagem apropriada se a entrada não for um número válido.

**27. Armazenamento Local (localStorage)**

**Exemplo Resolvido: Salvando e Carregando Preferências do Usuário**

**Problema:** Salve uma preferência de tema (ex: "dark" ou "light") no localStorage e, ao recarregar a página, carregue essa preferência.

*// Salvando uma preferência*

localStorage.setItem("temaPreferido", "dark");

console.log("Tema salvo:", localStorage.getItem("temaPreferido")); *// Saída: Tema salvo: dark*

*// Carregando uma preferência*

**const** temaSalvo = localStorage.getItem("temaPreferido");

**if** (temaSalvo) {

console.log(`O tema preferido do usuário é: **${**temaSalvo**}**`);

*// Aqui você aplicaria o tema à sua página*

} **else** {

console.log("Nenhuma preferência de tema encontrada.");

}

*// Removendo uma preferência (opcional)*

*// localStorage.removeItem("temaPreferido");*

**Explicação:** localStorage permite armazenar pares chave-valor de strings no navegador do usuário, persistindo mesmo após o fechamento da aba/navegador. setItem() salva um valor, getItem() recupera e removeItem() remove. É útil para dados não sensíveis que precisam persistir entre sessões.

**Seu Desafio: Contador de Visitas**

**Problema:** Crie um contador de visitas para uma página. Cada vez que a página é carregada, o contador deve ser incrementado e exibido no console. O valor do contador deve ser persistente, ou seja, não deve ser resetado ao fechar e abrir o navegador. (Dica: lembre-se que localStorage armazena strings, então você precisará converter para número ao ler e para string ao salvar).

**28. Expressões Regulares (RegEx)**

**Exemplo Resolvido: Validando Formato de CEP**

**Problema:** Verifique se uma string corresponde ao formato de CEP brasileiro (XXXXX XXX).

**const** cep1 = "12345-678";

**const** cep2 = "98765432";

**const** cep3 = "abcde-fgh";

*// Expressão regular para CEP: 5 dígitos, hífen, 3 dígitos*

**const** regexCEP = /^\d{5}-\d{3}$/;

console.log(`'**${**cep1**}**' é um CEP válido?`, regexCEP.test(cep1)); *// Saída: '12345-678' é um CEP válido? true*

console.log(`'**${**cep2**}**' é um CEP válido?`, regexCEP.test(cep2)); *// Saída: '98765432' é um CEP válido? false*

console.log(`'**${**cep3**}**' é um CEP válido?`, regexCEP.test(cep3)); *// Saída: 'abcde-fgh' é um CEP válido? false*

**Explicação:** Expressões regulares são padrões usados para encontrar combinações de caracteres em strings. ^ e $ marcam o início e o fim da string, respectivamente. \d corresponde a um dígito, {n} especifica o número de repetições. O método test() de uma RegEx retorna true se a string corresponder ao padrão.

**Seu Desafio: Validando Formato de Telefone**

**Problema:** Crie uma expressão regular para validar um formato de telefone simples, como (XX) XXXX-XXXX ou (XX) XXXXX-XXXX . Crie uma função que receba uma string e use essa RegEx para verificar se é um telefone válido. Teste com alguns exemplos.

**29. Manipulação de JSON**

**Exemplo Resolvido: Convertendo Objeto para JSON e Vice-versa**

**Problema:** Converta um objeto JavaScript para uma string JSON e, em seguida, converta essa string JSON de volta para um objeto JavaScript.

**const** usuario = {

nome: "Alice",

idade: 29,

email: "alice@example.com",

interesses: ["programação", "leitura"]

};

*// Convertendo objeto JavaScript para string JSON*

**const** usuarioJSON = JSON.stringify(usuario, **null**, 2);

console.log("Objeto como string JSON:", usuarioJSON);

*/\**

*Saída esperada:*

*Objeto como string JSON: {*

*"nome": "Alice",*

*"idade": 29,*

*"email": "alice@example.com",*

*"interesses": [*

*"programação",*

*"leitura"*

*]*

*}*

*\*/*

*// Convertendo string JSON de volta para objeto JavaScript*

**const** usuarioObjeto = JSON.parse(usuarioJSON);

console.log("String JSON como objeto JavaScript:", usuarioObjeto); *// Saída esperada: String JSON como objeto JavaScript: { nome: 'Alice', idade: 29, email: 'alice@example.com', interesses: [ 'programação', 'leitura' ] }*

**Explicação:** JSON.stringify() converte um valor JavaScript (geralmente um objeto ou array) em uma string JSON. O segundo argumento ( null ) e o terceiro ( 2 ) são opcionais e servem para formatar a saída JSON com indentação para melhor legibilidade. JSON.parse() faz o processo inverso, convertendo uma string JSON em um objeto ou array JavaScript.

**Seu Desafio: Salvando e Carregando Lista de Tarefas no localStorage**

**Problema:** Crie uma lista de tarefas (array de strings). Implemente duas funções: salvarTarefas(lista) que salva a lista no localStorage como uma string JSON, e carregarTarefas() que carrega a string JSON do localStorage e a converte de volta para um array. Teste salvando e carregando a lista.

**30. Web Workers (Conceito e Uso Básico)**

**Exemplo Resolvido: Executando um Cálculo Pesado em Segundo Plano**

**Problema:** Simule um cálculo demorado em um Web Worker para evitar travar a interface do usuário.

*// main.js (arquivo principal)*

**const** resultadoElement = document.createElement('p');

document.body.appendChild(resultadoElement);

**const** worker = **new** Worker('worker.js');

console.log('Iniciando cálculo pesado...');

resultadoElement.textContent = 'Calculando...';

*// Envia uma mensagem para o worker iniciar o cálculo*

worker.postMessage({ operacao: 'calcularSomaPesada', limite: 1000000000 });

*// Ouve mensagens do worker*

worker.onmessage = **function**(event) {

console.log('Mensagem recebida do worker:', event.data);

resultadoElement.textContent = `Resultado do cálculo:

**${**event.data.resultado**}**`; *// Saída: Resultado do cálculo: 500000000500000000* };

worker.onerror = **function**(error) {

console.error('Erro no worker:', error);

resultadoElement.textContent = 'Erro no cálculo.';

};

console.log('Interface do usuário continua responsiva.');

*// Você pode adicionar um botão ou outra interação aqui para provar que a UI não travou.*

*// worker.js (arquivo do Web Worker)*

self.onmessage = **function**(event) {

**const** { operacao, limite } = event.data;

**if** (operacao === 'calcularSomaPesada') {

**let** soma = 0;

**for** (**let** i = 1; i <= limite; i++) {

soma += i;

}

*// Envia o resultado de volta para o script principal*

self.postMessage({ resultado: soma });

}

};

**Explicação:** Web Workers permitem que scripts sejam executados em threads em segundo plano, separadas da thread principal da interface do usuário. Isso evita que operações demoradas congelem a página. A comunicação entre o script principal e o worker é feita via postMessage() e onmessage .

**Seu Desafio: Filtragem de Dados em Segundo Plano**

**Problema:** Suponha que você tenha um array muito grande de objetos (ex: 100.000 objetos de produtos) e precise filtrá-los com base em um critério (ex: categoria ). Crie um Web Worker para realizar essa filtragem em segundo plano. O script principal deve enviar o array e o critério para o worker, e o worker deve retornar o array filtrado. Exiba o resultado no console do script principal.