

Javascript



```
<script type="text/javascript">
  switch (new Date().getDay()) {
    case 6:
      text = "Friday";
      break;
    case 0:
      text = "Sunday";
      break;
    default:
      text = "Choose Your Day";
  }
</script>
```

Crie um plano de estudos que misture diferentes tópicos ou habilidades relacionados a javascript puro para me ajudar a desenvolver um entendimento mais robusto e facilitar a conexão entre os diferentes itens"

Claro! Aqui está um plano de estudos que mistura diferentes tópicos e habilidades relacionados ao JavaScript puro para ajudar a desenvolver um entendimento mais robusto e facilitar a conexão entre eles. O plano é dividido em diferentes áreas e você pode ajustá-lo de acordo com suas necessidades e conhecimentos prévios:

- 1 Introdução ao JavaScript:
 - 2 Sintaxe básica: variáveis, tipos de dados, operadores, condicionais e loops.
 - 3 Funções: declaração, parâmetros, argumentos e retorno.
 - 4 Objetos: propriedades, métodos e manipulação de objetos.
- 5 Manipulação do DOM:
 - 6 Selecionar elementos: usar seletores para obter elementos no DOM.
 - 7 Manipulação de elementos: alterar conteúdo, estilos, atributos e classes.
 - 8 Eventos: lidar com eventos do usuário, como clique, envio de formulário e hover.
- 9 Assincronismo:
 - 10 Callbacks: entender e usar funções de retorno de chamada.
 - 11 Promises: trabalhar com promessas para lidar com código assíncrono.
 - 12 Async/await: simplificar o código assíncrono usando async/await.
- 13 APIs e Requisições HTTP:
 - 14 Conceitos básicos de HTTP: GET, POST, PUT, DELETE.
 - 15 XMLHttpRequest: fazer requisições assíncronas a um servidor.
 - 16 Fetch API: usar a API Fetch para realizar requisições HTTP.
 - 17 Trabalhar com APIs de terceiros: consumir dados de APIs externas.
- 18 Manipulação de dados:
 - 19 JSON: entender a estrutura de dados JSON e como manipulá-la em JavaScript.
 - 20 Armazenamento local: usar o localStorage e sessionStorage para armazenar dados no navegador.
- 21 Testes e Depuração:
 - 22 Console: usar console.log() para depurar e exibir informações.
 - 23 Testes unitários: escrever e executar testes para garantir a qualidade do código.
- 24 Otimização e Performance:
 - 25 Boas práticas de codificação: escrever código limpo e legível.
 - 26 Otimização de desempenho: identificar gargalos e melhorar o desempenho do código.
 - 27 Ferramentas de desenvolvedor: utilizar ferramentas para analisar e otimizar o código.
- 28 Frameworks e bibliotecas:

- 29 Introdução ao React ou Vue.js: aprender os conceitos básicos de um framework JavaScript popular.
- 30 Manipulação do estado: entender como gerenciar o estado do aplicativo em um framework.
- 31 Projetos práticos:
 - 32 Aplicar os conhecimentos adquiridos em projetos pessoais.
 - 33 Participar de desafios de programação ou competições de codificação.

Lembre-se de praticar regularmente, experimentar com exemplos e projetos, e buscar recursos adicionais, como documentação oficial e tutoriais, para aprofundar-se nos tópicos conforme necessário. Boa sorte em sua jornada de aprendizado do JavaScript!

Monte o tópico 1

Certamente! Aqui está o tópico 1 do plano de estudos, que aborda a introdução ao JavaScript:

- 1 Introdução ao JavaScript:
 - a. Sintaxe básica:
 - 2 Variáveis: declaração, atribuição de valores e tipos de dados (números, strings, booleanos, arrays, objetos).
 - 3 Operadores: aritméticos (+, -, *, /), de atribuição (=, +=, -=, *=, /=), de comparação (==, ===, !=, !==, <, >, <=, >=), lógicos (&&, ||, !) e ternário (?).
 - 4 Condicionais: if, else if, else, switch case.
 - 5 Loops: for, while, do-while.
 - b. Funções:
 - 6 Declaração de função: criação de funções com a palavra-chave "function".
 - 7 Parâmetros e argumentos: definição de parâmetros em uma função e passagem de argumentos.
 - 8 Retorno de valores: uso da palavra-chave "return" para retornar valores de uma função.
 - c. Objetos:
 - 9 Propriedades: criação e acesso a propriedades de um objeto.
 - 10 Métodos: adição de métodos a objetos para executar ações.
 - 11 Manipulação de objetos: criação, atualização e exclusão de propriedades.

Recursos recomendados:

- 12 Documentação do JavaScript na MDN: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide>
- 13 Livro "Eloquent JavaScript" de Marijn Haverbeke: <https://eloquentjavascript.net/>
- 14 Curso online gratuito "JavaScript Basics" na plataforma freeCodeCamp: <https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/basic-javascript/>

Lembre-se de praticar os conceitos apresentados com exemplos de código e resolvendo exercícios para solidificar seu entendimento.

Mostre códigos de exemplo

Claro! Aqui estão alguns exemplos de código para ilustrar os conceitos abordados no tópico 1:

❶ Exemplo de declaração de variáveis e uso de operadores:

```
// Declaração de variáveis
let nome = 'João';
const idade = 25;
var salario = 3000;

// Uso de operadores
let soma = idade + 5;
let multiplicacao = salario * 12;
let ehMaior = idade > 18;
let resultado = (idade >= 18 && salario > 2000) ? 'Elegível' : 'Não elegível';

console.log(nome); // Saída: João
console.log(soma); // Saída: 30
console.log(multiplicacao); // Saída: 36000
console.log(ehMaior); // Saída: true
console.log(resultado); // Saída: Elegível
```

❶ Exemplo de uso de condicionais e loops:

```
// Condicional if/else
let numero = 10;

if (numero > 0) {
  console.log('O número é positivo');
} else if (numero < 0) {
  console.log('O número é negativo');
} else {
  console.log('O número é zero');
}

// Loop for
for (let i = 0; i < 5; i++) {
  console.log(i);
}

// Loop while
let contador = 0;
while (contador < 3) {
  console.log('Contador:', contador);
  contador++;
}
```

❶ Exemplo de declaração de função e retorno de valores:

```
// Declaração de função
function saudacao(nome) {
  return 'Olá, ' + nome + '!';
}

// Chamada da função e exibição do retorno
let mensagem = saudacao('Maria');
console.log(mensagem); // Saída: Olá, Maria!
```

❶ Exemplo de criação e manipulação de objetos:

```
// Criação de objeto
let pessoa = {
  nome: 'Carlos',
  idade: 30,
  profissao: 'Engenheiro',
};

// Acesso às propriedades do objeto
console.log(pessoa.nome); // Saída: Carlos
console.log(pessoa.idade); // Saída: 30

// Adição de método ao objeto
pessoa.apresentar = function() {
  console.log('Olá, meu nome é ' + this.nome + ' e tenho ' + this.idade + ' anos.');
```

Esses são apenas exemplos básicos para ilustrar os conceitos. Lembre-se de experimentar e praticar esses códigos, bem como criar seus próprios exemplos e exercícios para consolidar seu aprendizado.

Monte o tópico 2 e mostre exemplos

Claro! Aqui está o tópico 2 do plano de estudos, que aborda a manipulação do DOM (Document Object Model):

❶ Manipulação do DOM:

a. Selecionar elementos:

- ❷ Usar seletores para obter elementos no DOM.
- ❸ Utilizar métodos como `getElementById`, `getElementsByClassName`, `getElementsByTagName`, `querySelector`, `querySelectorAll` para selecionar elementos.

b. Manipulação de elementos:

- ❹ Alterar o conteúdo de elementos: modificar texto e HTML interno.
- ❺ Manipular estilos: modificar propriedades de estilo como cor, tamanho, visibilidade.
- ❻ Atualizar atributos: modificar atributos como `src`, `href`, `class`, `id`.
- ❼ Manipular classes: adicionar, remover e alternar classes em elementos.

c. Eventos:

- 8 Lidar com eventos do usuário, como clique, envio de formulário, hover.
- 9 Registrar manipuladores de eventos usando os métodos `addEventListener` e `on[evento]`.
- 10 Acessar informações do evento, como alvo, tecla pressionada, posição do cursor.

Recursos recomendados:

- 11 Documentação do JavaScript na MDN sobre manipulação do DOM: https://developer.mozilla.org/pt-BR/docs/Web/API/Document_Object_Model/Introduction
- 12 Curso online gratuito "JavaScript and the DOM" na plataforma Udacity: <https://www.udacity.com/course/javascript-and-the-dom--ud117>

Aqui estão alguns exemplos de código para ilustrar a manipulação do DOM:

- 1 Exemplo de seleção de elementos e alteração de conteúdo:

```
<!DOCTYPE html>
<html>
  <body>
    <h1 id="titulo">Título Original</h1>
    <p class="paragrafo">Este é um parágrafo.</p>

    <script>
      // Selecionar elemento pelo id
      let titulo = document.getElementById('titulo');
      console.log(titulo.textContent); // Saída: Título Original

      // Selecionar elementos pela classe
      let paragrafos = document.getElementsByClassName('paragrafo');
      console.log(paragrafos.length); // Saída: 1

      // Alterar conteúdo de um elemento
      titulo.textContent = 'Novo Título';
      paragrafos[0].textContent = 'Este é um novo parágrafo.';
    </script>
  </body>
</html>
```

- 1 Exemplo de manipulação de estilos e atributos:

```
<!DOCTYPE html>
<html>
  <body>
    <h1 id="titulo">Título Original</h1>
    

    <script>
      // Alterar estilo de um elemento
      let titulo = document.getElementById('titulo');
      titulo.style.color = 'blue';
      titulo.style.fontSize = '24px';

      // Atualizar atributos de um elemento
      let imagem = document.getElementById('imagem');
      imagem.src = 'nova-imagem.jpg';
      imagem.alt = 'Nova Imagem';
    </script>
  </body>
</html>
```

1 Exemplo de manipulação de classes e eventos:

```
<!DOCTYPE html>
<html>
  <body>
    <button id="btn">Clique Aqui</button>

    <script>
      // Manipular classes de um elemento
      let btn = document.getElementById('btn');
      btn.classList.add('destaque');
      btn.classList.remove('ativo');

      // Registrar manipulador de evento
      btn.addEventListener('click', function() {
```

Continue

Claro! Aqui está a continuação do exemplo de manipulação de classes e eventos:

```
<!DOCTYPE html>
<html>
  <body>
    <button id="btn">Clique Aqui</button>

    <script>
      // Manipular classes de um elemento
      let btn = document.getElementById('btn');
      btn.classList.add('destaque');
      btn.classList.remove('ativo');

      // Registrar manipulador de evento
      btn.addEventListener('click', function() {
        console.log('Botão clicado!');
      });
    </script>
  </body>
</html>
```

Neste exemplo, o código adiciona a classe "destaque" ao elemento de botão (btn) e remove a classe "ativo". Em seguida, é registrado um manipulador de evento para o evento de clique no botão. Quando o botão é clicado, a função de callback é executada e exibe a mensagem "Botão clicado!" no console.

Esses são apenas exemplos básicos para ilustrar a manipulação do DOM. Lembre-se de experimentar e praticar esses conceitos em seus próprios projetos, explorando diferentes seletores, estilos, atributos e eventos.

Claro! Aqui está o tópico 3 do plano de estudos, que aborda o assincronismo em JavaScript:

1 Assincronismo:

a. Callbacks:

- 2 Entender o conceito de callbacks como funções de retorno de chamada.
- 3 Utilizar callbacks para executar ações assíncronas e lidar com resultados.

b. Promises:

- 4 Compreender o uso de promises para tratar operações assíncronas.
- 5 Utilizar o construtor `Promise` para criar uma promise.
- 6 Encadear métodos `then()` e `catch()` para manipular o resultado ou erro de uma promise.

c. Async/await:

- 7 Utilizar a sintaxe `async/await` para simplificar o código assíncrono.
- 8 Utilizar a palavra-chave `async` antes de uma função para indicar que ela é assíncrona.
- 9 Utilizar a palavra-chave `await` antes de uma promise para aguardar a resolução da mesma.

Recursos recomendados:

- 10 Documentação do JavaScript na MDN sobre promises: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Promise
- 11 Documentação do JavaScript na MDN sobre async/await: https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Asynchronous/Async_await
- 12 Curso online gratuito "Asynchronous JavaScript" na plataforma Udacity: <https://www.udacity.com/course/asynchronous-javascript--ud109>

Aqui estão alguns exemplos de código para ilustrar o uso de callbacks, promises e async/await:

1 Exemplo de uso de callback:

```
function fetchData(callback) {
  setTimeout(function() {
    const data = 'Dados da API';
    callback(data);
  }, 2000);
}

function processarDados(dados) {
  console.log('Dados processados:', dados);
}

fetchData(processarDados);
```

Neste exemplo, a função `fetchData` simula uma operação assíncrona que retorna dados de uma API após 2 segundos. Essa função recebe um callback como argumento e, quando os dados estão prontos, chama o callback passando os dados como parâmetro. A função `processarDados` é usada como o callback para processar os dados recebidos.

1 Exemplo de uso de promises:


```
function fetchData() {
  return new Promise(function(resolve, reject) {
    setTimeout(function() {
      const data = 'Dados da API';
      if (data) {
        resolve(data);
      } else {
        reject('Erro ao obter os dados');
      }
    }, 2000);
  });
}

fetchData()
  .then(function(dados) {
    console.log('Dados recebidos:', dados);
  })
  .catch(function(erro) {
    console.log('Erro:', erro);
  });
```

Neste exemplo, a função `fetchData` retorna uma promise que envolve a operação assíncrona. Se os dados forem obtidos com sucesso, a promise é resolvida com os dados. Caso contrário, é rejeitada com uma mensagem de erro. Em seguida, o método `then()` é encadeado para manipular o caso de sucesso, enquanto o método `catch()` é usado para lidar com erros.

❶ Exemplo de uso de `async/await`:

```
function fetchData() {
  return new Promise(function(resolve, reject) {
    setTimeout(function() {
      const data =
```

Continue

Claro! Aqui está a continuação do exemplo de uso de `async/await`:

```
function fetchData() {
  return new Promise(function(resolve, reject) {
    setTimeout(function() {
      const data = 'Dados da API';
      if (data) {
        resolve(data);
      } else {
        reject('Erro ao obter os dados');
      }
    }, 2000);
  });
}

async function obterDados() {
  try {
    const dados = await fetchData();
    console.log('Dados recebidos:', dados);
  } catch (erro) {
```

```
        console.log('Erro:', erro);
    }
}

obterDados();
```

Neste exemplo, a função `fetchData` retorna uma promise, assim como no exemplo anterior. No entanto, a função `obterDados` é definida como uma função assíncrona usando a palavra-chave `async`. Dentro dessa função, utilizamos a palavra-chave `await` antes da chamada a `fetchData()` para aguardar a resolução da promise. Em seguida, o código continua a ser executado após a resolução da promise, permitindo que os dados sejam processados.

Lembre-se de que o uso de `async/await` requer que a função seja declarada como assíncrona (`async`), e a chamada a funções que retornam promises seja precedida por `await`. Isso permite escrever código assíncrono de forma mais síncrona e legível.

Esses exemplos ilustram o uso de callbacks, promises e `async/await` para lidar com operações assíncronas em JavaScript. Pratique esses conceitos em seus projetos e experimente diferentes casos de uso para obter uma compreensão mais sólida do assincronismo em JavaScript.

Tópico 4 agora

Com certeza! Aqui está o tópico 4 do plano de estudos, que aborda manipulação de arrays em JavaScript:

1 Manipulação de Arrays:

a. Criação e inicialização de arrays:

- 2 Declarar e inicializar arrays.
- 3 Utilizar a notação de colchetes ou o construtor `Array()` para criar arrays.

b. Acesso e modificação de elementos:

- 4 Acessar elementos de um array usando índices.
- 5 Modificar elementos de um array atribuindo um novo valor a um índice específico.

c. Iteração de arrays:

- 6 Utilizar loops como `for` e `forEach` para iterar sobre os elementos de um array.
- 7 Acessar e manipular cada elemento durante a iteração.

d. Métodos de array:

- 8 Utilizar métodos como `push`, `pop`, `shift`, `unshift`, `splice` para adicionar, remover ou modificar elementos em um array.
- 9 Utilizar métodos como `concat`, `slice`, `join`, `indexOf`, `find`, `filter`, `map`, `reduce` para realizar operações comuns em arrays.

Recursos recomendados:

- 10 Documentação do JavaScript na MDN sobre arrays: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array
- 11 Curso online gratuito "JavaScript Arrays" na plataforma freeCodeCamp: <https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/basic-javascript/>

Aqui estão alguns exemplos de código para ilustrar a manipulação de arrays:

❶ Exemplo de criação e acesso de elementos de um array:

```
// Declarar e inicializar um array
let frutas = ['maçã', 'banana', 'laranja'];

// Acessar elementos por índice
console.log(frutas[0]); // Saída: maçã
console.log(frutas[1]); // Saída: banana

// Modificar um elemento
frutas[2] = 'abacaxi';
console.log(frutas); // Saída: ['maçã', 'banana', 'abacaxi']
```

❶ Exemplo de iteração de um array usando o loop `for`:

```
let numeros = [1, 2, 3, 4, 5];

for (let i = 0; i < numeros.length; i++) {
  console.log(numeros[i]);
}
```

❶ Exemplo de uso de métodos de array:

```
let numeros = [1, 2, 3, 4, 5];

// Adicionar elementos no final do array
numeros.push(6);
console.log(numeros); // Saída: [1, 2, 3, 4, 5, 6]

// Remover o último elemento do array
numeros.pop();
console.log(numeros); // Saída: [1, 2, 3, 4, 5]

// Adicionar elementos no início do array
numeros.unshift(0);
console.log(numeros); // Saída: [0, 1, 2, 3, 4, 5]

// Remover o primeiro elemento do array
numeros.shift();
console.log(numeros); // Saída: [1, 2, 3, 4, 5]

// Remover elementos do array usando splice
numeros.splice(2, 2);
console.log(numeros); // Saída: [
```

Continue

Claro! Aqui está a continuação do exemplo de uso de métodos de array:

```
let numeros = [1, 2, 3, 4, 5];

// Remover elementos do array usando splice
numeros.splice(2, 2);
console.log(numeros); // Saída: [1, 2, 5]

// Concatenar arrays
let maisNumeros = [3, 4, 5];
let todosNumeros = numeros.concat(maisNumeros);
console.log(todosNumeros); // Saída: [1, 2, 5, 3, 4, 5]

// Filtrar elementos com base em uma condição
let numerosPares = todosNumeros.filter(function(numero) {
  return numero % 2 === 0;
});
console.log(numerosPares); // Saída: [2, 4]

// Mapear elementos para uma nova forma
let numerosDobrados = todosNumeros.map(function(numero) {
  return numero * 2;
});
console.log(numerosDobrados); // Saída: [2, 4, 10, 6, 8, 10]

// Reduzir elementos a um único valor
let somaTotal = todosNumeros.reduce(function(accumulador, numero) {
  return acumulador + numero;
}, 0);
console.log(somaTotal); // Saída: 20
```

Esses exemplos ilustram diferentes aspectos da manipulação de arrays em JavaScript. Lembre-se de explorar a documentação e praticar o uso desses métodos para se familiarizar com as várias operações disponíveis para manipular arrays de forma eficiente e concisa.

À medida que você avança nos estudos, lembre-se de aplicar esses conceitos em projetos reais para fortalecer sua compreensão e habilidades em manipulação de arrays em JavaScript.

Tópico 5

Com certeza! Aqui está o tópico 5 do plano de estudos, que aborda funções em JavaScript:

1 Funções:

a. Declaração de funções:

- 2 Declarar funções utilizando a palavra-chave `function`.
- 3 Definir parâmetros para uma função.

b. Chamada de funções:

- 4 Invocar funções utilizando a notação de parênteses e passando argumentos, se necessário.

c. Retorno de valores:

- 5 Utilizar a palavra-chave `return` para retornar um valor de uma função.
- 6 Compreender o conceito de valores de retorno e como eles são utilizados.

d. Escopo de variáveis:

- 7 Compreender os conceitos de escopo global e escopo local.
- 8 Utilizar variáveis locais e globais em diferentes partes do código.

e. Funções anônimas:

- 9 Utilizar funções anônimas como callbacks ou expressões de função.

Recursos recomendados:

- 10 Documentação do JavaScript na MDN sobre funções: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Funcoes>
- 11 Curso online gratuito "JavaScript Functions" na plataforma Codecademy: <https://www.codecademy.com/learn/introduction-to-javascript>

Aqui estão alguns exemplos de código para ilustrar o uso de funções em JavaScript:

- 1 Exemplo de declaração e chamada de função:

```
// Declaração de função
function saudacao() {
  console.log('Olá, seja bem-vindo!');
}

// Chamada de função
saudacao(); // Saída: Olá, seja bem-vindo!
```

- 1 Exemplo de função com parâmetros e retorno de valor:

```
// Função que recebe dois parâmetros e retorna a soma
function somar(a, b) {
  return a + b;
}

// Chamada da função e uso do valor de retorno
let resultado = somar(3, 5);
console.log(resultado); // Saída: 8
```

- 1 Exemplo de escopo de variáveis:

```
let nome = 'João'; // Variável global

function saudacao() {
  let mensagem = 'Olá, ' + nome; // Variável local
  console.log(mensagem);
}

saudacao(); // Saída: Olá, João
console.log(mensagem); // Erro: mensagem is not defined
```

- 1 Exemplo de função anônima como callback:

```
let numeros = [1, 2, 3, 4, 5];

// Função anônima como callback para o método forEach
numeros.forEach(function(numero) {
```

```
console.log(numero * 2);  
});
```

Neste exemplo, uma função anônima é passada como callback para o método `forEach` de um array. A função é executada para cada elemento do array, multiplicando o número por 2 e exibindo o resultado no console.

Lembre-se de praticar a criação e o uso de funções em diferentes contextos para aprimorar sua compreensão e familiaridade com esse importante conceito em JavaScript.

Tópico 6

Com certeza! Aqui está o tópico 6 do plano de estudos, que aborda objetos em JavaScript:

1 Objetos:

a. Criação de objetos:

- 2 Utilizar a notação de objeto literal para criar objetos.
- 3 Definir propriedades e valores para um objeto.

b. Acesso e modificação de propriedades:

- 4 Acessar propriedades de um objeto utilizando a notação de ponto ou colchetes.
- 5 Modificar propriedades de um objeto atribuindo um novo valor.

c. Métodos de objeto:

- 6 Definir métodos dentro de um objeto.
- 7 Acessar e invocar métodos de um objeto.

d. Herança e prototipagem:

- 8 Compreender o conceito de herança em JavaScript.
- 9 Utilizar a prototipagem para criar relações de herança entre objetos.

e. Uso de "this":

- 10 Compreender o uso da palavra-chave "this" para se referir ao objeto atual.
- 11 Utilizar "this" dentro de métodos para acessar propriedades e métodos do objeto.

Recursos recomendados:

- 12 Documentação do JavaScript na MDN sobre objetos: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Working_with_Objects
- 13 Curso online gratuito "JavaScript Objects" na plataforma Codecademy: <https://www.codecademy.com/learn/introduction-to-javascript>

Aqui estão alguns exemplos de código para ilustrar o uso de objetos em JavaScript:

- 1 Exemplo de criação e acesso de propriedades em um objeto:

```
// Criação de um objeto usando a notação de objeto literal
let pessoa = {
  nome: 'João',
  idade: 30,
  profissao: 'Desenvolvedor'
};

// Acesso às propriedades do objeto usando a notação de ponto
console.log(pessoa.nome); // Saída: João
console.log(pessoa.idade); // Saída: 30

// Modificação de uma propriedade
pessoa.idade = 31;
console.log(pessoa.idade); // Saída: 31
```

❶ Exemplo de definição de método em um objeto:

```
let pessoa = {
  nome: 'João',
  idade: 30,
  profissao: 'Desenvolvedor',
  saudacao: function() {
    console.log('Olá, meu nome é ' + this.nome + '. Tenho ' + this.idade + ' anos.');
```

❶ Exemplo de herança e prototipagem em objetos:

```
// Objeto pai
let animal = {
  comer: function() {
    console.log('O animal está comendo.');
```

Neste exemplo, o objeto `gato` é criado usando `Object.create(animal)`, o que estabelece

Continue

Neste exemplo, o objeto `gato` é criado usando `Object.create(animal)`, o que estabelece uma relação de herança entre o objeto `gato` e o objeto `animal`. Isso significa que o objeto `gato` herda as propriedades e métodos do objeto `animal`.

Lembre-se de praticar a criação, o acesso e a modificação de propriedades em objetos, bem como a definição de métodos e a compreensão do conceito de herança e prototipagem em JavaScript. Isso ajudará a fortalecer seu entendimento sobre objetos e como usá-los de forma eficaz em seus projetos.

Tópico 7

Com certeza! Aqui está o tópico 7 do plano de estudos, que aborda manipulação do DOM em JavaScript:

1 Manipulação do DOM:

a. Introdução ao DOM:

- 2 Compreender o que é o DOM (Modelo de Objeto de Documento) e sua estrutura hierárquica.
- 3 Entender a representação do HTML como uma árvore de elementos.

b. Acesso a elementos do DOM:

- 4 Utilizar métodos como `getElementById`, `querySelector` e `querySelectorAll` para selecionar elementos do DOM.
- 5 Acessar e modificar atributos e propriedades de elementos.

c. Manipulação de elementos:

- 6 Adicionar, remover e modificar elementos no DOM.
- 7 Manipular classes CSS de elementos.

d. Manipulação de eventos:

- 8 Registrar e responder a eventos como cliques, teclas pressionadas e submissão de formulários.
- 9 Utilizar manipuladores de eventos para executar ações específicas.

e. Traversing do DOM:

- 10 Navegar pelo DOM usando métodos como `parentNode`, `childNodes`, `nextSibling`, `previousSibling`, entre outros.

Recursos recomendados:

- 11 Documentação do JavaScript na MDN sobre o DOM: https://developer.mozilla.org/pt-BR/docs/Web/API/Document_Object_Model/Introduction
- 12 Curso online gratuito "JavaScript and the DOM" na plataforma freeCodeCamp: <https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/#javascript-and-the-dom>

Aqui estão alguns exemplos de código para ilustrar a manipulação do DOM em JavaScript:

1 Exemplo de acesso a elementos do DOM:

```
// Selecionar um elemento pelo ID
let meuElemento = document.getElementById('meu-id');

// Selecionar um elemento pela classe
let elementosDaClasse = document.getElementsByClassName('minha-classe');
```



```
// Selecionar um elemento pelo seletor CSS
let seletor = document.querySelector('.seletor-css');
let seletores = document.querySelectorAll('.seletor-css');
```

❶ Exemplo de manipulação de elementos do DOM:

```
// Criar um novo elemento
let novoElemento = document.createElement('div');

// Adicionar o novo elemento ao DOM
document.body.appendChild(novoElemento);

// Remover um elemento do DOM
let elementoParaRemover = document.getElementById('remover');
elementoParaRemover.parentNode.removeChild(elementoParaRemover);

// Modificar atributos e propriedades de elementos
let meuElemento = document.getElementById('meu-id');
meuElemento.setAttribute('class', 'nova-classe');
meuElemento.style.color = 'red';
```

❶ Exemplo de manipulação de eventos:

```
// Registrar um manipulador de evento para um clique
let meuBotao = document.getElementById('meu-botao');
meuBotao.addEventListener('click', function() {
  console.log('Botão clicado!');
});

// Registrar um manipulador de evento para uma tecla pressionada
document.addEventListener('keydown', function(event) {
  console.log('Tecla pressionada: ' + event.key);
});

// Registrar um manipulador de evento para a submissão de um formulário
let meuFormulario = document.getElementById('meu-formulario');
meuFormulario.addEventListener('submit', function(event) {
  event.preventDefault();
  console.log('Formulário enviado!');
});
```

Lembre-se de praticar a manipulação do DOM, experimentando diferentes seletores, métodos

Tópico 8

Claro! Aqui está o tópico 8 do plano de estudos, que aborda assincronismo em JavaScript:

❶ Assincronismo:

a. Callbacks:

❷ Compreender o conceito de callbacks e como eles são usados para lidar com tarefas assíncronas.

- 3 Utilizar callbacks em funções assíncronas para lidar com o resultado ou tratamento de erros.

b. Promises:

- 4 Entender o conceito de Promises e como elas simplificam o tratamento de tarefas assíncronas.
- 5 Utilizar Promises para lidar com operações assíncronas e encadear múltiplas operações.

c. Async/await:

- 6 Utilizar a sintaxe `async/await` para escrever código assíncrono de forma síncrona.
- 7 Compreender como o `async/await` simplifica o tratamento de Promises e melhora a legibilidade do código.

d. Manipulação de erros em operações assíncronas:

- 8 Lidar com erros em operações assíncronas utilizando `try/catch` e tratamento de exceções.

Recursos recomendados:

- 9 Documentação do JavaScript na MDN sobre Promises: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Promise
- 10 Curso online gratuito "JavaScript Promises" na plataforma Udacity: <https://www.udacity.com/course/javascript-promises--ud898>

Aqui estão alguns exemplos de código para ilustrar o assincronismo em JavaScript:

- 1 Exemplo de callback:

```
function obterDados(callback) {
  // Simulação de uma tarefa assíncrona
  setTimeout(function() {
    const dados = 'Dados obtidos!';
    callback(null, dados); // Passando null como erro e os dados como resultado
  }, 2000);
}

obterDados(function(erro, dados) {
  if (erro) {
    console.error(erro);
  } else {
    console.log(dados);
  }
});
```

- 1 Exemplo de Promise:

```
function obterDados() {
  return new Promise(function(resolve, reject) {
    // Simulação de uma tarefa assíncrona
    setTimeout(function() {
      const dados = 'Dados obtidos!';
      resolve(dados); // Resolvendo a Promise com os dados
      // reject(new Error('Ocorreu um erro!')); // Rejeitando a Promise com um erro
    }, 2000);
  });
}

obterDados()
  .then(function(dados) {
    console.log(dados);
  })
  .catch(function(erro) {
    console.error(erro);
  });
```

1 Exemplo de async/await:

```
function obterDados() {  
  return new Promise(function(resolve, reject) {  
    setTimeout(function() {  
      const dados = 'Dados obtidos!';  
      resolve(dados);  
    }, 2000);  
  });  
}  
  
async function main() {  
  try {  
    const dados = await obterDados();  
    console.log(dados);  
  } catch (erro) {  
    console.error(erro);  
  }  
}  
  
main();
```

Lembre-se de praticar o uso de callbacks, Promises e async/await para lidar com operações assíncronas. Isso permitirá que você desenvolva um código mais eficiente e legível ao lidar com tarefas assíncronas em JavaScript.

Tópico 9

Claro! Aqui está o tópico 9 do plano de estudos, que aborda manipulação de arrays em JavaScript:

1 Manipulação de Arrays:

a. Criação de arrays:

- 2** Declarar e inicializar um array em JavaScript.
- 3** Utilizar a notação literal de arrays ou o construtor `Array()`.

b. Acesso a elementos de um array:

- 4** Acessar elementos individuais em um array utilizando índices.
- 5** Compreender a indexação baseada em zero dos arrays em JavaScript.

c. Modificação de arrays:

- 6** Adicionar elementos a um array utilizando os métodos `push()`, `unshift()` e a atribuição direta.
- 7** Remover elementos de um array utilizando os métodos `pop()`, `shift()` e `splice()`.
- 8** Modificar elementos de um array diretamente.

d. Percorrendo arrays:

- 9** Utilizar loops como `for` e `forEach()` para percorrer elementos de um array.
- 10** Utilizar métodos como `map()`, `filter()` e `reduce()` para transformar e manipular arrays.

e. Operações comuns em arrays:

- 11 Verificar o tamanho de um array utilizando a propriedade `length`.
- 12 Concatenar e combinar arrays utilizando os métodos `concat()` e `splice()`.
- 13 Ordenar elementos de um array utilizando o método `sort()`.

Recursos recomendados:

- 14 Documentação do JavaScript na MDN sobre arrays: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array
- 15 Curso online gratuito "JavaScript Arrays" na plataforma Codecademy: <https://www.codecademy.com/learn/introduction-to-javascript>

Aqui estão alguns exemplos de código para ilustrar a manipulação de arrays em JavaScript:

- 1 Exemplo de criação e acesso a elementos de um array:

```
// Criação de um array com a notação literal
let frutas = ['maçã', 'banana', 'laranja'];

// Acesso a elementos individuais do array utilizando índices
console.log(frutas[0]); // Saída: maçã
console.log(frutas[1]); // Saída: banana
console.log(frutas[2]); // Saída: laranja
```

- 1 Exemplo de modificação de um array:

```
let frutas = ['maçã', 'banana', 'laranja'];

// Adicionar elementos a um array
frutas.push('morango'); // Adiciona 'morango' ao final do array
frutas.unshift('abacaxi'); // Adiciona 'abacaxi' no início do array

// Remover elementos de um array
frutas.pop(); // Remove o último elemento do array ('morango')
frutas.shift(); // Remove o primeiro elemento do array ('abacaxi')

// Modificar elementos de um array
frutas[1] = 'pera'; // Modifica o elemento no índice 1 para 'pera'
```

- 1 Exemplo de percorrer e manipular um array:

```
let numeros = [1, 2, 3, 4, 5];

// Utilizando um loop 'for' para percorrer os elementos do array
for (let i = 0; i < numeros.length; i++) {
  console.log(numeros[i]);
}

// Utilizando o método 'forEach()' para percorrer os elementos do array
numeros.forEach(function(numero) {
```

```
console.log(numero);
});

// Utilizando o método 'map()' para transformar um array
let dobrados = numeros.map(function(numero) {
  return numero * 2;
});

console.log(dobrados); // Saída: [2, 4, 6, 8, 10]

// Utilizando o método 'filter()' para filtrar elementos de um array
let pares = numeros.filter(function(numero) {
  return numero % 2 === 0;
});

console.log(pares); // Saída: [2, 4]

// Utilizando o método 'reduce()' para reduzir um array a um único valor
let soma = numeros.reduce(function(accumulador, numero) {
  return acumulador + numero;
}, 0);

console.log(soma); // Saída: 15
```

Lembre-se de praticar a criação, modificação e acesso a elementos de arrays, bem como a utilização de métodos como `forEach()`, `map()`, `filter()` e `reduce()` para manipular e transformar arrays. Isso ajudará a fortalecer sua compreensão sobre manipulação de arrays em JavaScript.

Grave esses tópicos lano de estudos JavaScript

User

Crie um plano de estudos que misture diferentes tópicos ou habilidades relacionados a javascript puro para me ajudar a desenvolver um entendimento mais robusto e facilitar a conexão entre os diferentes itens"

ChatGPT

Claro! Aqui está um plano de estudos que mistura diferentes tópicos e habilidades relacionados ao JavaScript puro para ajudar a desenvolver um entendimento mais robusto e facilitar a conexão entre eles. O plano é dividido em diferentes áreas e você pode ajustá-lo de acordo com suas necessidades e conhecimentos prévios:

Introdução ao JavaScript:

Sintaxe básica: variáveis, tipos de dados, operadores, condicionais e loops.

Funções: declaração, parâmetros, argumentos e retorno.

Objetos: propriedades, métodos e manipulação de objetos.

Manipulação do DOM:

Selecionar elementos: usar seletores para obter elementos no DOM.

Manipulação de elementos: alterar conteúdo, estilos, atributos e classes.

Eventos: lidar com eventos do usuário, como clique, envio de formulário e hover.

Assincronismo:

Callbacks: entender e usar funções de retorno de chamada.

Promises: trabalhar com promessas para lidar com código assíncrono.

Async/await: simplificar o código assíncrono usando async/await.

APIs e Requisições HTTP:

Conceitos básicos de HTTP: GET, POST, PUT, DELETE.

XMLHttpRequest: fazer requisições assíncronas a um servidor.

Fetch API: usar a API Fetch para realizar requisições HTTP.

Trabalhar com APIs de terceiros: consumir dados de APIs externas.

Manipulação de dados:

JSON: entender a estrutura de dados JSON e como manipulá-la em JavaScript.

Armazenamento local: usar o localStorage e sessionStorage para armazenar dados no navegador.

Testes e Depuração:

Console: usar console.log() para depurar e exibir informações.

Testes unitários: escrever e executar testes para garantir a qualidade do código.

Otimização e Performance:

Boas práticas de codificação: escrever código limpo e legível.

Otimização de desempenho: identificar gargalos e melhorar o desempenho do código.

Ferramentas de desenvolvedor: utilizar ferramentas para analisar e otimizar o código.

Frameworks e bibliotecas:

Introdução ao React ou Vue.js: aprender os conceitos básicos de um framework JavaScript popular.

Manipulação do estado: entender como gerenciar o estado do aplicativo em um framework.

Projetos práticos:

Aplicar os conhecimentos adquiridos em projetos pessoais.

Participar de desafios de programação ou competições de codificação.

Você pode organizar seu plano de estudos em etapas semanais ou mensais, definindo quantos tópicos deseja cobrir em cada período. Lembre-se de reservar tempo para revisar e praticar os conceitos aprendidos. Além disso, aproveite os recursos recomendados e outros materiais disponíveis online, como tutoriais, artigos e documentações oficiais.

Aqui está um exemplo de um plano de estudos de 8 semanas para você começar:

Semana 1:

- Introdução ao JavaScript: Sintaxe básica, variáveis, tipos de dados, operadores.
- Funções: Declaração, parâmetros, argumentos e retorno.

Semana 2:

- Manipulação do DOM: Selecionar elementos, manipulação de conteúdo e estilos.
- Eventos: Lidar com eventos do usuário.

Semana 3:

- Assincronismo: Callbacks.
- APIs e Requisições HTTP: Conceitos básicos de HTTP.

Semana 4:

- Assincronismo: Promises.
- Manipulação de dados: JSON.

Semana 5:

- Assincronismo: Async/await.
- APIs e Requisições HTTP: XMLHttpRequest.

Semana 6:

- Manipulação de arrays: Criação, acesso e modificação de arrays.
- Manipulação de dados: Armazenamento local.

Semana 7:

- Testes e Depuração: Console, testes unitários.
- Otimização e Performance: Boas práticas de codificação.

Semana 8:

- Frameworks e bibliotecas: Introdução ao React ou Vue.js.
- Projetos práticos: Aplicar os conhecimentos em um projeto pessoal.

Ajuste o plano conforme necessário, adicionando mais semanas ou dividindo os tópicos em períodos menores. Lembre-se de dedicar tempo regularmente para praticar e revisar os conceitos aprendidos, pois a prática é fundamental para fortalecer seu entendimento e habilidades em JavaScript.

Disserte sobre com exemplos:
APIs e Requisições HTTP:

Conceitos básicos de HTTP: GET, POST, PUT, DELETE.
XMLHttpRequest: fazer requisições assíncronas a um servidor.
Fetch API: usar a API Fetch para realizar requisições HTTP.
Trabalhar com APIs de terceiros: consumir dados de APIs externas.
Manipulação de dados:

JSON: entender a estrutura de dados JSON e como manipulá-la em JavaScript.
Armazenamento local: usar o localStorage e sessionStorage para armazenar dados no navegador.
Testes e Depuração:

Console: usar console.log() para depurar e exibir informações.
Testes unitários: escrever e executar testes para garantir a qualidade do código.

Otimização e Performance:

Boas práticas de codificação: escrever código limpo e legível.
Otimização de desempenho: identificar gargalos e melhorar o desempenho do código.
Ferramentas de desenvolvedor: utilizar ferramentas para analisar e otimizar o código.
Frameworks e bibliotecas:

Introdução ao React ou Vue.js: aprender os conceitos básicos de um framework JavaScript popular.
Manipulação do estado: entender como gerenciar o estado do aplicativo em um framework.
Projetos práticos:

Aplicar os conhecimentos adquiridos em projetos pessoais.
Participar de desafios de programação ou competições de codificação.

APIs e Requisições HTTP:

As APIs (Application Programming Interfaces) desempenham um papel fundamental no desenvolvimento de aplicativos modernos. Elas permitem a comunicação entre diferentes sistemas e permitem que os desenvolvedores acessem e manipulem dados de serviços externos. Vamos explorar alguns conceitos e exemplos relacionados a APIs e requisições HTTP.

❶ Conceitos básicos de HTTP: GET, POST, PUT, DELETE O protocolo HTTP (Hypertext Transfer Protocol) é a base das comunicações na web. Ele define diferentes métodos para realizar operações em recursos. Os principais métodos são:

- GET: Usado para recuperar dados de um recurso específico.
- POST: Utilizado para enviar dados para serem processados por um recurso.
- PUT: Utilizado para atualizar um recurso existente com novos dados.
- DELETE: Usado para remover um recurso específico.

Esses métodos são essenciais para interagir com APIs e realizar operações de leitura, gravação, atualização e exclusão de dados.

❶ XMLHttpRequest: fazer requisições assíncronas a um servidor O objeto XMLHttpRequest é uma API antiga, mas ainda amplamente utilizada, para fazer requisições assíncronas a um servidor. Ele permite enviar solicitações HTTP para um servidor e manipular a resposta recebida. Aqui está um exemplo de uso do XMLHttpRequest para realizar uma solicitação GET:

```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'https://api.example.com/data', true);

xhr.onload = function() {
  if (xhr.status === 200) {
    var response = JSON.parse(xhr.responseText);
    console.log(response);
  }
};

xhr.send();
```

❶ Fetch API: usar a API Fetch para realizar requisições HTTP A Fetch API é uma API moderna que fornece uma interface mais poderosa e flexível para fazer requisições HTTP. Ela é baseada em promessas e oferece uma sintaxe mais simples e concisa. Aqui está um exemplo de uso da Fetch API para fazer uma solicitação GET:


```
fetch('https://api.example.com/data')
  .then(function(response) {
    if (response.ok) {
      return response.json();
    } else {
      throw new Error('Erro na solicitação.');
```

- 1 Trabalhar com APIs de terceiros: consumir dados de APIs externas Uma das principais vantagens das APIs é a capacidade de consumir dados de serviços externos. Muitos serviços disponibilizam APIs públicas que permitem que os desenvolvedores acessem e utilizem seus dados. Por exemplo, você pode consumir a API do Twitter para exibir tweets em seu aplicativo ou a API do Google Maps para exibir mapas interativos. Cada API possui sua própria documentação e método de autenticação, portanto, é importante ler a documentação fornecida pelo serviço antes de começar a usá-la.

Manipulação de dados:

- 1 JSON: entender a estrutura de dados JSON e como manipulá-la em JavaScript JSON (JavaScript Object Notation) é um formato de dados leve e amplamente utilizado para trocar dados entre um servidor e um cliente. Ele é fácil de ler e escrever para humanos e fácil de analisar e gerar para máquinas. No

Continue

JavaScript, podemos usar os métodos `JSON.stringify()` e `JSON.parse()` para converter objetos JavaScript em strings JSON e vice-versa. Aqui está um exemplo:

```
// Converter objeto JavaScript em string JSON
var pessoa = { nome: 'João', idade: 25, cidade: 'São Paulo' };
var pessoaJSON = JSON.stringify(pessoa);
console.log(pessoaJSON); // Saída: {"nome":"João","idade":25,"cidade":"São Paulo"}

// Converter string JSON em objeto JavaScript
var pessoaObjeto = JSON.parse(pessoaJSON);
console.log(pessoaObjeto); // Saída: {nome: 'João', idade: 25, cidade: 'São Paulo'}
```

- 1 Armazenamento local: usar o `localStorage` e `sessionStorage` para armazenar dados no navegador O armazenamento local no navegador permite que você armazene dados persistentes no dispositivo do usuário. O `localStorage` e o `sessionStorage` são objetos JavaScript que fornecem métodos simples para armazenar e recuperar dados. A diferença entre eles está na duração dos dados armazenados: o `localStorage` mantém os dados mesmo quando o navegador é fechado, enquanto o `sessionStorage` armazena os dados apenas durante a sessão de navegação atual. Aqui está um exemplo de uso do `localStorage`:

```
// Armazenar dados no localStorage
localStorage.setItem('nome', 'João');
localStorage.setItem('idade', '25');
```

```
// Recuperar dados do localStorage
var nome = localStorage.getItem('nome');
var idade = localStorage.getItem('idade');

console.log(nome, idade); // Saída: João 25
```

1 Testes e Depuração:

- Console: usar `console.log()` para depurar e exibir informações. O `console.log()` é uma ferramenta essencial para depurar seu código JavaScript. Ele permite exibir mensagens e informações no console do navegador, facilitando a verificação de valores de variáveis, o rastreamento do fluxo do programa e a identificação de erros. Aqui está um exemplo simples:

```
var nome = 'João';
var idade = 25;

console.log(nome, idade); // Saída: João 25
```

- Testes unitários: escrever e executar testes para garantir a qualidade do código. Os testes unitários são uma prática importante no desenvolvimento de software, pois garantem que partes específicas do código funcionem corretamente. Existem várias estruturas e bibliotecas de testes disponíveis para JavaScript, como Jest, Mocha e Jasmine, que facilitam a escrita e execução de testes automatizados. Aqui está um exemplo simples usando o Jest:

```
// Função para somar dois números
function somar(a, b) {
  return a + b;
}

// Teste para verificar se a função retorna o resultado esperado
test('Somar 2 + 3 deve retornar 5', () => {
  expect(somar(2, 3)).toBe(5);
});
```

1 Otimização e Performance:

- Boas práticas de codificação: escrever código limpo e legível. Seguir boas práticas de codificação é fundamental para manter o código organizado, legível e de fácil manutenção. Alguns princípios importantes incluem usar nomes descritivos para variáveis e funções, evitar duplicação de código, manter a consistência na formatação e usar comentários para explicar partes complexas

Continue

as do código. Aqui estão algumas boas práticas de codificação em JavaScript:

- Use nomes descritivos para variáveis, funções e classes.
- Evite duplicação de código, extraia partes repetidas para funções reutilizáveis.
- Mantenha a consistência na formatação do código, como indentação e espaçamento.
- Comente seu código para explicar a lógica complexa, tornando-o mais compreensível para outros desenvolvedores.

- Siga as convenções de estilo de código, como o guia de estilo do JavaScript do Airbnb ou do Google.
- Otimização de desempenho: identificar gargalos e melhorar o desempenho do código.
A otimização de desempenho é importante para garantir que o código JavaScript execute de maneira eficiente. Alguns pontos a serem considerados são:
 - Minimizar o número de solicitações de rede e o tamanho dos arquivos enviados para o navegador.
 - Otimizar loops e operações de manipulação de arrays, evitando processamentos desnecessários.
 - Usar recursos nativos e eficientes do JavaScript, como métodos de array (por exemplo, `map()`, `filter()`, `reduce()`).
 - Evitar o uso excessivo de operações custosas, como manipulação direta do DOM em loops.
 - Utilizar técnicas de caching para evitar cálculos repetidos ou solicitações desnecessárias ao servidor.
- Ferramentas de desenvolvedor: utilizar ferramentas para analisar e otimizar o código.
Existem várias ferramentas de desenvolvedor disponíveis para JavaScript que podem ajudar na análise e otimização do código. Algumas das mais comuns incluem:
 - Chrome DevTools: um conjunto de ferramentas embutidas no navegador Google Chrome que permite depurar, perfilar e monitorar o desempenho do código JavaScript.
 - Lighthouse: uma ferramenta automatizada de análise e otimização para web apps, que pode identificar problemas de desempenho, acessibilidade, SEO, entre outros.
 - ESLint: uma ferramenta de linting que ajuda a identificar erros e problemas de estilo no código JavaScript, seguindo as regras definidas no arquivo de configuração.
 - Webpack: um bundler de módulos JavaScript que pode otimizar e empacotar o código para melhor desempenho e eficiência.

Frameworks e bibliotecas:

- Introdução ao React ou Vue.js: aprender os conceitos básicos de um framework JavaScript popular.
O React e o Vue.js são frameworks JavaScript populares para a construção de interfaces de usuário interativas e reativas. Eles adotam abordagens diferentes, mas ambos oferecem uma maneira eficiente de criar componentes reutilizáveis e dinâmicos. Aprender um desses frameworks pode abrir muitas oportunidades de desenvolvimento web moderno.
- Manipulação do estado: entender como gerenciar o estado do aplicativo em um framework.
Gerenciar o estado do aplicativo é fundamental para o desenvolvimento de aplicativos JavaScript complexos. Tanto o React quanto o Vue.js oferecem soluções para gerenciamento de estado, como o React's `useState` e `useContext` hooks, e o `Vuex` no Vue.js. Aprender a gerenciar o estado do aplicativo é essencial para criar aplicativos escaláveis e fáceis de manter.

Projetos práticos:

Aplicar

Continue

os conhecimentos adquiridos em projetos pessoais é uma ótima maneira de consolidar o aprendizado e aprimorar suas habilidades de programação. Aqui estão algumas ideias de projetos práticos nos quais você pode aplicar os conceitos mencionados:

- 1 Criar um aplicativo de lista de tarefas: Utilize um framework como React ou Vue.js para construir um aplicativo de lista de tarefas no qual os usuários possam adicionar, remover e marcar tarefas como concluídas. Você pode armazenar os

dados localmente usando o localStorage ou até mesmo conectar-se a uma API para salvar as tarefas em um servidor.

- 2 Desenvolver um aplicativo de previsão do tempo: Consuma dados de uma API de previsão do tempo, como a OpenWeatherMap, e crie um aplicativo que exiba as condições climáticas atuais e previsões futuras. Use as requisições HTTP e manipulação de dados em JSON para obter e exibir as informações relevantes.
- 3 Criar um aplicativo de catálogo de filmes: Consuma dados de uma API de filmes, como o The Movie Database (TMDb), e crie um aplicativo no qual os usuários possam pesquisar e explorar filmes. Utilize recursos como requisições HTTP, manipulação de dados JSON e exibição de informações detalhadas sobre os filmes.
- 4 Construir um aplicativo de blog pessoal: Desenvolva um aplicativo de blog usando um framework como React ou Vue.js. Os usuários podem criar, editar e excluir postagens, e você pode armazenar os dados no localStorage ou se conectar a uma API para salvar as postagens em um servidor.
- 5 Participar de desafios de programação ou competições de codificação: Desafie-se participando de competições de codificação, como o Codeforces ou o HackerRank. Essas plataformas oferecem uma variedade de problemas e desafios de programação que podem ajudar a aprimorar suas habilidades de resolução de problemas e algoritmos.

Lembre-se de adaptar essas ideias de projetos de acordo com seus interesses e nível de habilidade. O objetivo é aplicar os conceitos aprendidos em cenários reais e ganhar experiência prática no desenvolvimento de aplicativos.

Todo com js puro

Desculpe pela confusão anterior. Aqui estão algumas ideias de projetos práticos usando apenas JavaScript puro:

- 1 Jogo da Velha: Crie um jogo da velha interativo em que dois jogadores possam competir entre si. Use o JavaScript para manipular o estado do jogo, controlar a lógica das jogadas e exibir o tabuleiro.
- 2 Calculadora: Desenvolva uma calculadora simples que permita realizar operações matemáticas básicas, como adição, subtração, multiplicação e divisão. Use JavaScript para capturar os cliques nos botões e exibir o resultado.
- 3 Gerador de Senhas: Crie um gerador de senhas aleatórias que permita aos usuários especificar o comprimento e os critérios desejados, como letras maiúsculas, letras minúsculas, números e caracteres especiais. Utilize JavaScript para gerar a senha e exibi-la.
- 4 Lista de Tarefas: Implemente um aplicativo de lista de tarefas no qual os usuários possam adicionar, remover e marcar as tarefas como concluídas. Use JavaScript para manipular a lista de tarefas e atualizar a interface do usuário.
- 5 Relógio Digital: Crie um relógio digital que exiba a hora atual em tempo real. Use JavaScript para obter a hora atual do sistema e atualize a exibição a cada segundo.
- 6 Validador de Formulários: Desenvolva um validador de formulários que verifique se os campos foram preenchidos corretamente antes de enviar os dados. Use JavaScript para capturar os eventos de envio do formulário e validar os campos de acordo com as regras definidas.
- 7 Carrossel de Imagens: Crie um carrossel de imagens que exiba várias imagens em uma sequência animada. Use JavaScript para controlar a transição entre as imagens e permitir a navegação pelos slides.
- 8 Gerador de QR Code: Implemente um gerador de QR Code que permita aos usuários inserir um texto ou URL e gere o QR Code correspondente. Utilize JavaScript para gerar o código QR e exibi-lo na tela.
- 9 Cronômetro: Desenvolva um cronômetro com funcionalidades de iniciar, pausar e reiniciar. Use JavaScript para controlar o tempo e atualizar a exibição dos segundos, minutos e horas.

Essas são apenas algumas ideias para projetos práticos usando JavaScript puro. Você pode adaptá-las e expandi-las de acordo com suas preferências e necessidades. Lembre-se de que o objetivo é praticar o uso de JavaScript em projetos reais e

consolidar seus conhecimentos na linguagem.

Jogo da Velha: Crie um jogo da velha interativo em que dois jogadores possam competir entre si. Use o JavaScript para manipular o estado do jogo, controlar a lógica das jogadas e exibir o tabuleiro.

Certamente! Aqui está um exemplo de como você pode criar um jogo da velha interativo usando JavaScript puro:

HTML:

```
<div class="board">
  <div class="cell" onclick="makeMove(0)"></div>
  <div class="cell" onclick="makeMove(1)"></div>
  <div class="cell" onclick="makeMove(2)"></div>
  <div class="cell" onclick="makeMove(3)"></div>
  <div class="cell" onclick="makeMove(4)"></div>
  <div class="cell" onclick="makeMove(5)"></div>
  <div class="cell" onclick="makeMove(6)"></div>
  <div class="cell" onclick="makeMove(7)"></div>
  <div class="cell" onclick="makeMove(8)"></div>
</div>
<button onclick="resetGame()">Reiniciar</button>
```

CSS:

```
.board {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-gap: 5px;
  width: 300px;
  height: 300px;
  margin-bottom: 10px;
}

.cell {
  background-color: lightgray;
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: 40px;
  cursor: pointer;
}

button {
  font-size: 16px;
}
```

JavaScript:

```
let currentPlayer = 'X';
let gameActive = true;
let gameState = ['', '', '', '', '', '', '', '', ''];

const winningCombinations = [
```

```

[0, 1, 2], [3, 4, 5], [6, 7, 8], // horizontal
[0, 3, 6], [1, 4, 7], [2, 5, 8], // vertical
[0, 4, 8], [2, 4, 6] // diagonal
];

function makeMove(cellIndex) {
  if (gameState[cellIndex] === '' && gameActive) {
    gameState[cellIndex] = currentPlayer;
    document.getElementsByClassName('cell')[cellIndex].textContent = currentPlayer;
    if (checkWin()) {
      alert(`Jogador ${currentPlayer} venceu!`);
      gameActive = false;
    } else if (!gameState.includes('')) {
      alert("Empate!");
      gameActive = false;
    } else {
      currentPlayer = currentPlayer === 'X' ? 'O' : 'X';
    }
  }
}

function checkWin() {
  for (let i = 0; i < winningCombinations.length; i++) {
    const [a, b, c] = winningCombinations[i];
    if (gameState[a] === gameState[b] && gameState[b] === gameState[c] && gameState[a] !== '') {
      return true;
    }
  }
  return false;
}

function resetGame() {
  currentPlayer = 'X';
  gameActive = true;
  gameState = ['', '', '', '', '', '', '', '', ''];
  const cells = document.getElementsByClassName('cell');
  for (let i = 0; i < cells.length; i++) {
    cells[i].textContent = '';
  }
}

```

Neste exemplo, criamos uma estrutura de jogo da velha usando elementos HTML `<div>` para representar as células do tabuleiro. Cada célula possui um evento de clique que chama a função `makeMove()` passando o índice correspondente. A função `makeMove()` verifica se a

Continue

célula está vazia e o jogo está ativo antes de permitir que o jogador faça a jogada. Em seguida, atualiza o estado do jogo, exibindo o símbolo do jogador atual na célula correspondente. Em seguida, verifica se o jogador atual venceu o jogo ou se houve um empate. Se uma das condições for atendida, exibe uma mensagem de vitória ou empate e desativa o jogo.

A função `checkWin()` percorre todas as combinações possíveis de vitória e verifica se o jogador atual preencheu essas células.

A função `resetGame()` reinicia o jogo, redefinindo todas as variáveis para seus valores iniciais e limpando o conteúdo das células.

Para visualizar o jogo da velha em ação, você pode copiar e colar o código acima em um arquivo HTML e abri-lo em um navegador. A partir daí, você pode jogar clicando nas células do tabuleiro e reiniciar o jogo pressionando o botão "Reiniciar".

Este é apenas um exemplo básico de jogo da velha em JavaScript puro. Você pode personalizar o estilo, adicionar animações, implementar um modo para um jogador contra o computador e expandir as funcionalidades conforme desejado.

Calendário com opção de mudar o ano e mês

Aqui está um exemplo de um calendário interativo em JavaScript puro que permite ao usuário mudar o ano e o mês:

HTML:

```
<div class="calendar">
  <div class="header">
    <button onclick="previousMonth()">&lt;</button>
    <h2 id="month-year"></h2>
    <button onclick="nextMonth()">&gt;>/button>
  </div>
  <table id="calendar-table">
    <thead>
      <tr>
        <th>Domingo</th>
        <th>Segunda</th>
        <th>Terça</th>
        <th>Quarta</th>
        <th>Quinta</th>
        <th>Sexta</th>
        <th>Sábado</th>
      </tr>
    </thead>
    <tbody id="calendar-body"></tbody>
  </table>
</div>
```

CSS:

```
.calendar {
  width: 300px;
}

.header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 10px;
}

button {
  background: none;
  border: none;
  cursor: pointer;
  font-size: 16px;
}

#month-year {
  font-size: 18px;
}
```

```

table {
  width: 100%;
  border-collapse: collapse;
}

th, td {
  text-align: center;
  padding: 5px;
  border: 1px solid #ccc;
}

```

JavaScript:

```

let currentDate = new Date();
let currentMonth = currentDate.getMonth();
let currentYear = currentDate.getFullYear();

function generateCalendar() {
  const calendarBody = document.getElementById('calendar-body');
  const monthYearLabel = document.getElementById('month-year');

  monthYearLabel.textContent = getMonthName(currentMonth) + ' ' + currentYear;
  calendarBody.innerHTML = '';

  const firstDayOfMonth = new Date(currentYear, currentMonth, 1).getDay();
  const totalDaysInMonth = new Date(currentYear, currentMonth + 1, 0).getDate();

  let day = 1;

  for (let i = 0; i < 6; i++) {
    const row = document.createElement('tr');

    for (let j = 0; j < 7; j++) {
      if (i === 0 && j < firstDayOfMonth) {
        const cell = document.createElement('td');
        row.appendChild(cell);
      } else if (day > totalDaysInMonth) {
        break;
      } else {
        const cell = document.createElement('td');
        cell.textContent = day;
        row.appendChild(cell);
        day++;
      }
    }

    calendarBody.appendChild(row);
  }
}

function previousMonth() {
  currentMonth--;
  if (currentMonth < 0) {
    currentMonth = 11;
    currentYear--;
  }
  generateCalendar();
}

function nextMonth() {
  currentMonth++;
  if (currentMonth > 11) {
    currentMonth = 0;
    currentYear++;
  }
  generateCalendar();
}

```



```
function getMonthName(monthIndex) {  
  const monthNames = ['Janeiro', 'Fevereiro', 'Março', 'Abril', 'Maio', 'Junho', 'Julho', 'Agosto', 'Setembro', 'Outubro'];  
  return monthNames[monthIndex];  
}  
  
generateCalendar();
```

Neste exemplo, criamos um calendário simples que exibe o mês atual e permite que o usuário navegue para o mês anterior e próximo usando

Continue

os botões de navegação. A função `generateCalendar()` é responsável por gerar o calendário com base no mês e ano atuais. Ela calcula o primeiro dia do mês e o total de dias no mês usando o objeto `Date`, e então itera sobre as células da tabela para preenchê-las com os números dos dias correspondentes.

As funções `previousMonth()` e `nextMonth()` atualizam o valor do mês e ano atuais, levando em consideração o limite inferior e superior dos meses. Em seguida, chamam a função `generateCalendar()` para atualizar o calendário com o mês e ano atualizados.

A função `getMonthName()` retorna o nome do mês com base no índice do mês.

Você pode copiar e colar o código acima em um arquivo HTML e abri-lo em um navegador para ver o calendário em ação. Os botões de navegação permitirão que você alterne entre os meses e anos. O calendário será gerado dinamicamente com base nas datas selecionadas.

Você também pode personalizar o estilo CSS conforme desejado para tornar o calendário mais atraente. Além disso, é possível adicionar funcionalidades extras, como destaque do dia atual, eventos e interações adicionais com o calendário.

Espero que isso ajude você a criar um calendário interativo em JavaScript puro!

Calculadora com sen cos raiz potências etc

Aqui está um exemplo de uma calculadora básica em JavaScript puro que inclui funções para cálculos trigonométricos (seno, cosseno), raiz quadrada e potências:

HTML:

```
<div class="calculator">  
  <input type="text" id="result" readonly>  
  <div class="buttons">  
    <button onclick="clearResult()">C</button>  
    <button onclick="deleteLastChar()">&larr;</button>  
    <button onclick="appendChar('7')">7</button>
```

```

<button onclick="appendChar('8')">8</button>
<button onclick="appendChar('9')">9</button>
<button onclick="appendChar('/')">/</button>
<button onclick="appendChar('4')">4</button>
<button onclick="appendChar('5')">5</button>
<button onclick="appendChar('6')">6</button>
<button onclick="appendChar('*')">*</button>
<button onclick="appendChar('1')">1</button>
<button onclick="appendChar('2')">2</button>
<button onclick="appendChar('3')">3</button>
<button onclick="appendChar('-')">-</button>
<button onclick="appendChar('0')">0</button>
<button onclick="appendChar('.')">.</button>
<button onclick="calculate()">=</button>
<button onclick="appendChar('+')">+</button>
<button onclick="calculateTrigFunction('sin')">sin</button>
<button onclick="calculateTrigFunction('cos')">cos</button>
<button onclick="calculateSqrt()">sqrt</button>
<button onclick="calculatePower()">x^y</button>
</div>
</div>

```

CSS:

```

.calculator {
  width: 200px;
  margin: 0 auto;
  text-align: center;
}

#result {
  width: 100%;
  padding: 5px;
  margin-bottom: 10px;
}

.buttons {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  gap: 5px;
}

button {
  padding: 10px;
  font-size: 16px;
  background-color: #eee;
  border: none;
  cursor: pointer;
}

button:hover {
  background-color: #ccc;
}

```

JavaScript:

```

function clearResult() {
  document.getElementById('result').value = '';
}

function deleteLastChar() {
  let result = document.getElementById('result').value;
  result = result.slice(0, -1);
  document.getElementById('result').value = result;
}

```

```

function appendChar(char) {
  document.getElementById('result').value += char;
}

function calculate() {
  const result = document.getElementById('result').value;
  const calculatedResult = eval(result);
  document.getElementById('result').value = calculatedResult;
}

function calculateTrigFunction(func) {
  const result = document.getElementById('result').value;
  let calculatedResult;

  if (func === 'sin') {
    calculatedResult = Math.sin(eval(result));
  } else if (func === 'cos') {
    calculatedResult = Math.cos(eval(result));
  }

  document.getElementById('result').value = calculatedResult;
}

function calculateSqrt() {
  const result = document.getElementById('result').value;
  const calculatedResult = Math.sqrt(eval(result));
  document.getElementById('result').value = calculatedResult;
}

function calculatePower() {
  const result = document.getElementById('result').value;
  const parts = result.split('^');
  const base = eval

```

Continue

```

(parts[0]); const exponent = eval(parts[1]); const calculatedResult = Math.pow(base, exponent);
document.getElementById('result').value = calculatedResult; }

```

Neste exemplo, a calculadora consiste em um campo de texto onde o resultado é exibido e uma grade de botões que representam as operações matemáticas. As funções JavaScript são responsáveis por manipular as interações do usuário e realizar os cálculos correspondentes. A função `calculate()` avalia a expressão matemática presente no campo de texto usando a função `eval()` e exibe o resultado no campo de texto. As funções `calculateTrigFunction(func)`, `calculateSqrt()` e `calculatePower()` realizam os cálculos trigonométricos, raiz quadrada e potência, respectivamente. Ao copiar o código acima para um arquivo HTML e abri-lo em um navegador, você terá uma calculadora interativa com as funcionalidades mencionadas. Lembre-se de que a função `eval()` pode ser arriscada se o código-fonte do usuário for inserido diretamente na expressão.

Gerador de senha com opção de escolher só letras min maiusc caracteres e números ou todos e tamanho dela

Aqui está um exemplo de um gerador de senha em JavaScript puro que permite escolher o tipo de caracteres (letras minúsculas, letras maiúsculas, caracteres especiais e números) e o tamanho da senha:

HTML:

```
<div class="password-generator">
  <label for="password-length">Tamanho da senha:</label>
  <input type="number" id="password-length" min="1" value="8">
  <br>
  <label for="lowercase-checkbox">Letras minúsculas</label>
  <input type="checkbox" id="lowercase-checkbox" checked>
  <br>
  <label for="uppercase-checkbox">Letras maiúsculas</label>
  <input type="checkbox" id="uppercase-checkbox" checked>
  <br>
  <label for="special-chars-checkbox">Caracteres especiais</label>
  <input type="checkbox" id="special-chars-checkbox" checked>
  <br>
  <label for="numbers-checkbox">Números</label>
  <input type="checkbox" id="numbers-checkbox" checked>
  <br>
  <button onclick="generatePassword()">Gerar senha</button>
  <br>
  <input type="text" id="generated-password" readonly>
</div>
```

CSS:

```
.password-generator {
  width: 200px;
  margin: 0 auto;
  text-align: center;
}

input[type="number"] {
  width: 60px;
  margin-bottom: 10px;
}

button {
  margin-top: 10px;
  padding: 5px 10px;
  font-size: 16px;
  background-color: #eee;
  border: none;
  cursor: pointer;
}

#generated-password {
  margin-top: 10px;
  width: 100%;
  padding: 5px;
}
```

JavaScript:

```
function generatePassword() {
  const length = document.getElementById('password-length').value;
  const lowercase = document.getElementById('lowercase-checkbox').checked;
  const uppercase = document.getElementById('uppercase-checkbox').checked;
  const specialChars = document.getElementById('special-chars-checkbox').checked;
  const numbers = document.getElementById('numbers-checkbox').checked;

  const lowercaseChars = 'abcdefghijklmnopqrstuvwxyz';
  const uppercaseChars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
  const specialChars = '!@#$%^&*()';
  const numberChars = '0123456789';

  let allChars = '';
  let password = '';

  if (lowercase) {
    allChars += lowercaseChars;
    password += getRandomChar(lowercaseChars);
  }

  if (uppercase) {
    allChars += uppercaseChars;
    password += getRandomChar(uppercaseChars);
  }

  if (specialChars) {
    allChars += specialChars;
    password += getRandomChar(specialChars);
  }

  if (numbers) {
    allChars += numberChars;
    password += getRandomChar(numberChars);
  }

  for (let i = password.length; i < length; i++) {
    password += getRandomChar(allChars);
  }

  document.getElementById('generated-password').value = password;
}

function getRandomChar(characters) {
  const randomIndex = Math.floor(Math.random() * characters.length);
  return characters.charAt(randomIndex);
}
```

Neste exemplo, a página HTML contém opções para selecionar o tamanho da senha e os tipos de caracteres permitidos (letras minúsculas, letras maiúsculas, caracteres especiais e números). Ao clicar no botão "Gerar senha", a função `generatePassword()` é acionada.

A função `generatePassword()` rec

Continue

A função `generatePassword()` recupera os valores selecionados pelo usuário, como o tamanho da senha e os tipos de caracteres permitidos. Em seguida, ela constrói uma string `allChars` que contém todos os caracteres permitidos com base nas

opções selecionadas.

A função `getRandomChar(characters)` recebe uma string de caracteres e retorna um caractere aleatório dessa string. Ela é usada para selecionar caracteres aleatórios para a senha.

A função `generatePassword()` começa selecionando um caractere aleatório de cada tipo de caractere selecionado e adicionando-os à senha. Em seguida, ela itera até que a senha tenha o tamanho especificado, adicionando caracteres aleatórios da string `allChars` restante.

Finalmente, a senha gerada é exibida no campo de texto de saída.

Ao copiar o código acima para um arquivo HTML e abri-lo em um navegador, você terá um gerador de senha interativo com as opções de escolher os tipos de caracteres permitidos e o tamanho da senha.

Este é um exemplo básico e não inclui tratamento de erros ou validação de entrada. É recomendado implementar essas funcionalidades adicionais para tornar o gerador de senha mais robusto.

Certifique-se de que seu código esteja seguro ao lidar com senhas geradas. Este exemplo é apenas uma ilustração e não aborda considerações de segurança específicas.

Lista de tarefas

Aqui está um exemplo de uma lista de tarefas em JavaScript puro, onde você pode adicionar, marcar como concluídas e remover tarefas:

HTML:

```
<div class="task-list">
  <input type="text" id="task-input" placeholder="Adicionar uma nova tarefa">
  <button onclick="addTask()">Adicionar</button>
  <ul id="task-list-ul"></ul>
</div>
```

CSS:

```
.task-list {
  width: 300px;
  margin: 0 auto;
}

input[type="text"] {
  width: 200px;
  padding: 5px;
}

button {
  padding: 5px 10px;
  background-color: #eee;
  border: none;
  cursor: pointer;
}

ul {
  list-style-type: none;
  padding: 0;
}
```

```
li.completed {  
  text-decoration: line-through;  
}
```

JavaScript:

```
function addTask() {  
  const taskInput = document.getElementById('task-input');  
  const taskText = taskInput.value.trim();  
  
  if (taskText !== '') {  
    const taskList = document.getElementById('task-list-ul');  
    const taskItem = document.createElement('li');  
    const taskCheckbox = document.createElement('input');  
    taskCheckbox.type = 'checkbox';  
    taskCheckbox.addEventListener('change', toggleTask);  
    const taskLabel = document.createElement('label');  
    taskLabel.textContent = taskText;  
    const taskDeleteBtn = document.createElement('button');  
    taskDeleteBtn.textContent = 'Remover';  
    taskDeleteBtn.addEventListener('click', removeTask);  
  
    taskItem.appendChild(taskCheckbox);  
    taskItem.appendChild(taskLabel);  
    taskItem.appendChild(taskDeleteBtn);  
    taskList.appendChild(taskItem);  
  
    taskInput.value = '';  
  }  
}  
  
function toggleTask(event) {  
  const taskItem = event.target.parentNode;  
  taskItem.classList.toggle('completed');  
}  
  
function removeTask(event) {  
  const taskItem = event.target.parentNode;  
  const taskList = taskItem.parentNode;  
  taskList.removeChild(taskItem);  
}
```

Neste exemplo, a página HTML consiste em um campo de texto para adicionar uma nova tarefa, um botão para adicionar a tarefa à lista e uma lista não ordenada (ul) para exibir as tarefas. Cada tarefa é exibida como um item de lista (li) que contém um checkbox para marcar como concluída, um label para exibir o texto da tarefa e um botão para remover a tarefa da lista.

A função `addTask()` é acionada quando o botão "Adicionar" é clicado. Ela recupera o texto da tarefa do campo de texto, cria os elementos HTML correspondentes à tarefa e os adiciona à lista de tarefas.

A função `toggleTask()` é acionada quando o checkbox da tarefa é alterado. Ela adiciona ou remove a classe "completed" ao item da tarefa, para indicar se a tarefa está concluída ou não.

A função `removeTask()` é acionada quando o botão "Remover" é clicado. Ela remove o item da tarefa da lista.

Ao copiar o código acima para um arquivo HTML e abri-lo em um navegador, você terá uma lista de tarefas interativa onde você pode adicionar, marcar como concluídas e remover tarefas.

Este é um exemplo básico e não inclui recursos de armazenamento persistente ou outras funcionalidades avançadas. Você pode expandir esse exemplo adicionando funcionalidades como armazenamento local

Para adicionar funcionalidades de armazenamento persistente à lista de tarefas, você pode utilizar a API Web Storage, que permite armazenar dados no navegador. Neste exemplo, usaremos o `localStorage` para salvar as tarefas.

Atualize a função `addTask()` com o código abaixo:

```
function addTask() {
  const taskInput = document.getElementById('task-input');
  const taskText = taskInput.value.trim();

  if (taskText !== '') {
    const taskList = document.getElementById('task-list-ul');
    const taskItem = document.createElement('li');
    const taskCheckbox = document.createElement('input');
    taskCheckbox.type = 'checkbox';
    taskCheckbox.addEventListener('change', toggleTask);
    const taskLabel = document.createElement('label');
    taskLabel.textContent = taskText;
    taskDeleteBtn = document.createElement('button');
    taskDeleteBtn.textContent = 'Remover';
    taskDeleteBtn.addEventListener('click', removeTask);

    taskItem.appendChild(taskCheckbox);
    taskItem.appendChild(taskLabel);
    taskItem.appendChild(taskDeleteBtn);
    taskList.appendChild(taskItem);

    // Salvar tarefa no localStorage
    saveTask(taskText);

    taskInput.value = '';
  }
}

function saveTask(taskText) {
  let tasks = localStorage.getItem('tasks');

  if (tasks) {
    tasks = JSON.parse(tasks);
    tasks.push(taskText);
  } else {
    tasks = [taskText];
  }

  localStorage.setItem('tasks', JSON.stringify(tasks));
}
```

Adicione o seguinte código no final do script JavaScript:

```
// Carregar tarefas salvas no localStorage
window.addEventListener('DOMContentLoaded', loadTasks);

function loadTasks() {
  const tasks = localStorage.getItem('tasks');

  if (tasks) {
    const taskList = document.getElementById('task-list-ul');
    const tasksArray = JSON.parse(tasks);

    tasksArray.forEach((taskText) => {
      const taskItem = document.createElement('li');
      const taskCheckbox = document.createElement('input');
```



```

    taskCheckbox.type = 'checkbox';
    taskCheckbox.addEventListener('change', toggleTask);
    const taskLabel = document.createElement('label');
    taskLabel.textContent = taskText;
    const taskDeleteBtn = document.createElement('button');
    taskDeleteBtn.textContent = 'Remover';
    taskDeleteBtn.addEventListener('click', removeTask);

    taskItem.appendChild(taskCheckbox);
    taskItem.appendChild(taskLabel);
    taskItem.appendChild(taskDeleteBtn);
    taskList.appendChild(taskItem);
  });
}
}

```

Agora, sempre que uma nova tarefa for adicionada, ela será salva no localStorage utilizando a função `saveTask()`. Quando a página for carregada, a função `loadTasks()` será executada, recuperando as tarefas salvas no localStorage e exibindo-as na lista.

Dessa forma, as tarefas adicionadas persistirão mesmo se você atualizar a página ou fechar o navegador.

Este é apenas um exemplo básico de como adicionar armazenamento persistente às tarefas. Você pode explorar outras opções, como bancos de dados locais ou APIs de armazenamento online, para criar uma solução mais robusta e escalável.

Relógio digital, contador de tempo e cronômetro opção de escolher cada um

Aqui está um exemplo de um relógio digital, um contador de tempo regressivo e um cronômetro em JavaScript puro, com a opção de escolher entre cada um:

HTML:

```

<div class="clock">
  <h2>Relógio Digital</h2>
  <p id="clock-time"></p>
</div>

<div class="timer">
  <h2>Contador de Tempo Regressivo</h2>
  <div>
    <input type="number" id="timer-input" placeholder="Tempo em segundos">
    <button onclick="startTimer()">Iniciar</button>
    <button onclick="resetTimer()">Reiniciar</button>
  </div>
  <p id="timer-display"></p>
</div>

<div class="stopwatch">
  <h2>Cronômetro</h2>
  <div>
    <button onclick="startStopwatch()">Iniciar</button>
    <button onclick="stopStopwatch()">Parar</button>
    <button onclick="resetStopwatch()">Reiniciar</button>
  </div>
  <p id="stopwatch-display">00:00:00</p>
</div>

```

CSS:

```
.clock,
.timer,
.stopwatch {
  margin-bottom: 20px;
}

button {
  margin-right: 10px;
}
```

JavaScript:

```
// Relógio Digital
function startClock() {
  const clockTime = document.getElementById('clock-time');
  const currentTime = new Date().toLocaleTimeString();
  clockTime.textContent = currentTime;
  setTimeout(startClock, 1000);
}

// Contador de Tempo Regressivo
let timerInterval;
let timerSeconds;

function startTimer() {
  const timerInput = document.getElementById('timer-input');
  const timerDisplay = document.getElementById('timer-display');
  const seconds = parseInt(timerInput.value, 10);

  if (isNaN(seconds) || seconds <= 0) {
    return;
  }

  timerSeconds = seconds;
  timerInterval = setInterval(updateTimer, 1000);
  timerInput.disabled = true;
}

function updateTimer() {
  const timerDisplay = document.getElementById('timer-display');

  if (timerSeconds <= 0) {
    clearInterval(timerInterval);
    timerDisplay.textContent = 'Tempo encerrado';
    return;
  }

  const hours = Math.floor(timerSeconds / 3600);
  const minutes = Math.floor((timerSeconds % 3600) / 60);
  const seconds = timerSeconds % 60;

  timerDisplay.textContent = `${hours.toString().padStart(2, '0')}:${minutes
    .toString()
    .padStart(2, '0')}:${seconds.toString().padStart(2, '0')}`;

  timerSeconds--;
}

function resetTimer() {
  clearInterval(timerInterval);
  const timerInput = document.getElementById('timer-input');
  const timerDisplay = document.getElementById('timer-display');
  timerInput.disabled = false;
  timerInput.value = '';
  timerDisplay.textContent = '';
}
```

```

}

// Cronômetro
let stopwatchInterval;
let stopwatchSeconds;

function startStopwatch() {
  const stopwatchDisplay = document.getElementById('stopwatch-display');
  stopwatchSeconds = 0;
  stopwatchInterval = setInterval(updateStopwatch, 1000);
}

function updateStopwatch() {
  const stopwatchDisplay = document.getElementById('stopwatch-display');
  stopwatchSeconds++;

  const hours = Math.floor(stopwatchSeconds / 3600);
  const minutes = Math.floor((stopwatchSeconds % 3600) / 60);
  const seconds = stopwatchSeconds % 60;

  stopwatchDisplay.textContent = `${hours.toString().padStart(2, '

```

Continue

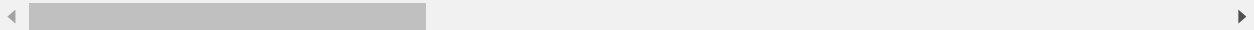
```
0')):${minutes.toString().padStart(2, '0')}:${seconds.toString().padStart(2, '0')}`; }
```

```
function stopStopwatch() { clearInterval(stopwatchInterval); }
```

```
function resetStopwatch() { clearInterval(stopwatchInterval); const stopwatchDisplay = document.getElementById('stopwatch-display'); stopwatchDisplay.textContent = '00:00:00'; }
```

```
// Iniciar o relógio digital ao carregar a página window.addEventListener('DOMContentLoaded', startClock);
```

Este exemplo apresenta um relógio digital que exibe a hora atual, um contador de tempo regressivo onde você pode in:



Carrossel de imagens

Aqui está um exemplo básico de um carrossel de imagens em JavaScript puro:

HTML:

```

<div class="carousel">
  <button class="carousel-button prev" onclick="prevSlide()">&lt;</button>
  <div class="carousel-images">
    
    
    
  </div>
  <button class="carousel-button next" onclick="nextSlide()">&gt;</button>
</div>

```

CSS:

```

.carousel {
  position: relative;
  width: 300px;
  margin: 0 auto;
}

.carousel-images {
  display: flex;
  overflow: hidden;
}

.carousel img {
  width: 100%;
}

.carousel-button {
  position: absolute;
  top: 50%;
  transform: translateY(-50%);
  background-color: transparent;
  border: none;
  color: #fff;
  font-size: 24px;
  cursor: pointer;
  outline: none;
}

.prev {
  left: 10px;
}

.next {
  right: 10px;
}

```

JavaScript:

```

let slideIndex = 0;
const slides = document.querySelectorAll('.carousel-images img');

function showSlide(index) {
  slides.forEach((slide) => {
    slide.style.display = 'none';
  });

  if (index >= slides.length) {
    slideIndex = 0;
  } else if (index < 0) {
    slideIndex = slides.length - 1;
  }

  slides[slideIndex].style.display = 'block';
}

```

```
function nextSlide() {
  slideIndex++;
  showSlide(slideIndex);
}

function prevSlide() {
  slideIndex--;
  showSlide(slideIndex);
}

showSlide(slideIndex);
```

Neste exemplo, temos um carrossel simples com três imagens. As setas "anterior" e "próxima" permitem navegar entre as imagens. O JavaScript controla a exibição dos slides, alterando o estilo de exibição para mostrar apenas o slide atual e ocultar os demais.

Você pode personalizar o HTML e o CSS de acordo com suas necessidades de design, adicionando mais imagens ou ajustando as dimensões do carrossel.

Lembre-se de substituir os nomes das imagens (`imagem1.jpg`, `imagem2.jpg`, `imagem3.jpg`) pelos caminhos corretos para suas próprias imagens.

Gerador de Qr code

Para criar um gerador de QR code em JavaScript puro, você pode utilizar a biblioteca QRious, que permite a geração de QR codes no navegador. Aqui está um exemplo básico:

HTML:

```
<div>
  <input type="text" id="qr-text" placeholder="Digite o texto ou URL">
  <button onclick="generateQRCode()">Gerar QR Code</button>
</div>
<div id="qr-code"></div>
```

JavaScript:

```
function generateQRCode() {
  const qrText = document.getElementById('qr-text').value;
  const qrCodeDiv = document.getElementById('qr-code');

  // Limpar conteúdo anterior do div
  qrCodeDiv.innerHTML = '';

  // Criar o objeto QRious
  const qr = new QRious({
    element: qrCodeDiv,
    value: qrText,
    size: 200
  });
}
```

Este exemplo possui um campo de texto onde você pode inserir um texto ou URL e um botão para gerar o QR code correspondente. Ao clicar no botão, o código chama a função `generateQRCode()`, que recupera o texto inserido e cria um novo objeto QRious com o valor fornecido. O QR code é exibido dentro do elemento `qr-code` com tamanho de 200 pixels.

Certifique-se de incluir a biblioteca QRious em seu projeto. Você pode baixar a biblioteca ou utilizar um link de CDN. Por exemplo, você pode adicionar a seguinte tag `<script>` no cabeçalho do documento HTML:

```
<script src="https://cdn.jsdelivr.net/npm/qrrious@4.0.2/qrrious.min.js"></script>
```

Lembre-se de verificar a documentação da biblioteca QRious para obter mais informações sobre as opções de personalização e recursos adicionais disponíveis.

Qr code com javascript puro

Para gerar um QR code usando JavaScript puro, você pode utilizar a biblioteca `QRCode.js`, que permite a criação de QR codes no navegador sem a necessidade de dependências adicionais. Aqui está um exemplo básico de como utilizar essa biblioteca:

HTML:

```
<div>
  <input type="text" id="qr-text" placeholder="Digite o texto ou URL">
  <button onclick="generateQRCode()">Gerar QR Code</button>
</div>
<div id="qr-code"></div>
```

JavaScript:

```
function generateQRCode() {
  const qrText = document.getElementById('qr-text').value;
  const qrCodeDiv = document.getElementById('qr-code');

  // Limpar conteúdo anterior do div
  qrCodeDiv.innerHTML = '';

  // Criar elemento de imagem para o QR code
  const qrCodeImg = document.createElement('img');

  // Configurar o tamanho e dados do QR code
  qrCodeImg.width = 200;
  qrCodeImg.height = 200;
  qrCodeImg.src = `https://api.qrserver.com/v1/create-qr-code/?data=${encodeURIComponent(qrText)}&size=200x200`;

  // Adicionar a imagem ao elemento div
  qrCodeDiv.appendChild(qrCodeImg);
}
```

Neste exemplo, há um campo de texto onde você pode inserir um texto ou URL e um botão para gerar o QR code correspondente. Ao clicar no botão, a função `generateQRCode()` é chamada, que recupera o texto inserido, cria um elemento de imagem e define sua largura, altura e src para a URL do serviço de geração de QR codes. O QR code é exibido dentro do elemento `qr-code`.

É importante observar que esse exemplo utiliza um serviço externo para gerar o QR code. No caso apresentado, é utilizado o serviço "QR Server" (<https://www.qrserver.com/>) através de sua API. O texto do QR code é passado como parâmetro na URL e o tamanho do QR code é definido como 200x200 pixels.

Certifique-se de incluir o código HTML e JavaScript em um arquivo HTML válido e abra-o em um navegador para testar o gerador de QR code.

Lembre-se de verificar a documentação da biblioteca QRCode.js para mais informações sobre personalização e recursos adicionais disponíveis.

Quiz com as perguntas em um arquivo Json em separado com 5 opções e ao ser clicado de a opção de corrigir na hora apresentando a resposta certa

Para criar um quiz com perguntas em um arquivo JSON separado, você pode seguir os passos abaixo:

- 1 Crie um arquivo JSON chamado `questions.json` com o seguinte formato:

```
{
  "questions": [
    {
      "question": "Qual é a capital do Brasil?",
      "options": ["Rio de Janeiro", "São Paulo", "Brasília", "Salvador", "Curitiba"],
      "answer": 2
    },
    {
      "question": "Qual é o maior planeta do Sistema Solar?",
      "options": ["Júpiter", "Terra", "Vênus", "Netuno", "Saturno"],
      "answer": 0
    },
    {
      "question": "Quem pintou a Mona Lisa?",
      "options": ["Leonardo da Vinci", "Pablo Picasso", "Vincent van Gogh", "Michelangelo", "Claude Monet"],
      "answer": 0
    },
    // Adicione mais perguntas aqui
  ]
}
```

- 1 Crie um arquivo HTML com o seguinte conteúdo:

```
<!DOCTYPE html>
<html>
<head>
  <title>Quiz</title>
  <style>
    #quiz {
      max-width: 500px;
      margin: 0 auto;
    }
  </style>
</head>
<body>
  <div id="quiz"></div>

  <script src="quiz.js"></script>
```

```
</body>
</html>
```

- 1 Crie um arquivo JavaScript chamado `quiz.js` com o seguinte código:

```
fetch('questions.json')
  .then(response => response.json())
  .then(data => {
    const quizContainer = document.getElementById('quiz');
    const questions = data.questions;
    let currentQuestion = 0;
    let score = 0;

    function showQuestion() {
      const questionData = questions[currentQuestion];
      const questionElement = document.createElement('div');
      questionElement.innerHTML = `
        <h2>${questionData.question}</h2>
        <ul>
          ${questionData.options.map((option, index) => `
            <li>
              <input type="radio" name="answer" value="${index}">
                ${option}
            </li>
          `).join('')}
        </ul>
        <button onclick="checkAnswer()">Verificar</button>
        <div id="result"></div>
      `;
      quizContainer.innerHTML = '';
      quizContainer.appendChild(questionElement);
    }

    function checkAnswer() {
      const selectedOption = document.querySelector('input[name="answer"]:checked');
      if (selectedOption) {
        const userAnswer = parseInt(selectedOption.value);
        const questionData = questions[currentQuestion];
        const resultElement = document.getElementById('result');

        if (userAnswer === questionData.answer) {
          resultElement.textContent = 'Resposta correta!';
          score++;
        } else {
          resultElement.textContent = `Resposta incorreta! A resposta correta é: ${questionData.options[questionData.answer]}`;
        }

        currentQuestion++;

        if (currentQuestion < questions.length) {
          showQuestion();
        } else {
          showResults();
        }
      }
    }

    function showResults() {
      quizContainer.innerHTML = `
        <h2>Quiz Finalizado!</h2>
        <p>Você acertou ${score} de ${questions.length} perguntas.</p>
        <button onclick="location.reload()">Reiniciar</button>
      `;
    }
  })
```



```
showQuestion();  
});
```

Neste exemplo, o arquivo `quiz.js` faz uma requisição para o arquivo

Continue

`questions.json` usando o método `fetch()` para obter os dados das perguntas. Em seguida, as perguntas são armazenadas na variável `questions` e as variáveis `currentQuestion` e `score` são inicializadas.

A função `showQuestion()` é responsável por exibir a pergunta atual, as opções de resposta e um botão para verificar a resposta. O evento de clique do botão chama a função `checkAnswer()`.

A função `checkAnswer()` verifica se o usuário selecionou uma opção de resposta. Se sim, a resposta é comparada com a resposta correta da pergunta atual. Se a resposta estiver correta, a variável `score` é incrementada. Em seguida, a variável `currentQuestion` é atualizada para avançar para a próxima pergunta. Se houver mais perguntas, a função `showQuestion()` é chamada novamente para exibir a próxima pergunta. Caso contrário, a função `showResults()` é chamada para exibir o resultado final do quiz.

A função `showResults()` exibe a pontuação final do usuário e um botão para reiniciar o quiz.

Certifique-se de que os arquivos `questions.json`, `quiz.js` e o arquivo HTML estejam na mesma pasta e abra o arquivo HTML em um navegador para testar o quiz.

Lembre-se de personalizar o arquivo JSON com suas próprias perguntas, opções de resposta e respostas corretas.

Mostre um json com 20 questões de javascript e a resposta pro quiz anterior

Aqui está um exemplo de um arquivo JSON com 20 questões de JavaScript e suas respostas para o quiz anterior:

```
{  
  "questions": [  
    {  
      "question": "Qual é a saída do seguinte código?\n\nconsole.log(2 + 2 + '2');",  
      "options": ["4", "22", "222", "NaN"],  
      "answer": 1  
    },  
    {  
      "question": "Qual é o resultado da expressão:\n\n3 * '3'",  
      "options": ["9", "'9'", "NaN", "TypeError"],  
      "answer": 2  
    },  
    {  
      "question": "Qual é a forma correta de escrever um comentário de uma linha em JavaScript?",  
      "options": ["// Comentário", "/* Comentário */", "<!-- Comentário -->", "# Comentário"],  
    }  
  ]  
}
```

```
"answer": 0
},
{
  "question": "Qual é a função utilizada para converter uma string em um número inteiro?",
  "options": ["parseInt()", "parseFloat()", "Number()", "Math.round()"],
  "answer": 0
},
{
  "question": "Qual é o operador de igualdade estrita em JavaScript?",
  "options": ["=", "==", "===", "!="],
  "answer": 2
},
{
  "question": "Qual é a função utilizada para obter o comprimento (número de caracteres) de uma string?",
  "options": ["length()", "size()", "count()", "sizeof()"],
  "answer": 0
},
{
  "question": "Qual é a sintaxe correta para declarar uma variável em JavaScript?",
  "options": ["var name;", "variable name;", "name = var;", "name = value;"],
  "answer": 0
},
{
  "question": "Qual é a função utilizada para exibir uma caixa de diálogo com uma mensagem para o usuário?",
  "options": ["console.log()", "alert()", "confirm()", "prompt()"],
  "answer": 1
},
{
  "question": "Qual é a saída do seguinte código?\n\nconsole.log(typeof null);",
  "options": ["null", "undefined", "object", "boolean"],
  "answer": 2
},
{
  "question": "Qual é o método utilizado para adicionar um elemento ao final de um array em JavaScript?",
  "options": ["push()", "add()", "concat()", "append()"],
  "answer": 0
},
{
  "question": "Qual é a forma correta de escrever um laço de repetição 'for' em JavaScript?",
  "options": ["for (var i = 0; i < 5; i++)", "for (i = 0; i < 5; i++)", "for (i < 5; i++)", "for (i = 0; i <= 5; i++)"],
  "answer": 0
},
{
  "question": "Qual é o método utilizado para remover o último elemento de um array em JavaScript?",
  "options": ["pop()", "remove()", "delete()", "slice()"],
  "answer": 0
},
{
  "question": "Qual é o operador lógico utilizado para negar uma expressão em JavaScript?",
  "options": ["&&", "||", "!", "==="],
  "answer": 2
},
{
  "question": "
```

Continue

```
{
  "question": "Qual é a função utilizada para arredondar um número para o inteiro mais próximo em JavaScript?",
  "options": ["round()", "floor()", "ceil()", "abs()"],
  "answer": 0
},
{
  "question": "Qual é a forma correta de escrever uma função em JavaScript?",
  "options": ["function = myFunction() {}", "function myFunction() {}", "myFunction() = function() {}", "myFunction = function() {}"],
  "answer": 1
},
{
  "question": "Qual é o método utilizado para converter um array em uma string em JavaScript?",
  "options": ["toString()", "join()", "concat()", "slice()"],
  "answer": 1
},
{
  "question": "Qual é o operador utilizado para concatenar strings em JavaScript?",
  "options": ["+", "-", "", "/"],
  "answer": 0
},
{
  "question": "Qual é o método utilizado para obter uma parte específica de uma string em JavaScript?",
  "options": ["substring()", "slice()", "substr()", "splice()"],
  "answer": 1
},
{
  "question": "Qual é a forma correta de escrever uma estrutura condicional 'if' em JavaScript?",
  "options": ["if (condition) { // code }", "{ if condition // code }", "if { condition // code }", "if { condition: // code }"],
  "answer": 0
},
{
  "question": "Qual é o método utilizado para remover um elemento específico de um array em JavaScript?",
  "options": ["remove()", "splice()", "pop()", "shift()"],
  "answer": 1
},
{
  "question": "Qual é o operador utilizado para atribuir um valor a uma variável em JavaScript?",
  "options": ["==", "=", "===", "+="],
  "answer": 1
},
{
  "question": "Qual é a função utilizada para arredondar um número para baixo em JavaScript?",
  "options": ["round()", "floor()", "ceil()", "abs()"],
  "answer": 1
},
{
  "question": "Qual é a sintaxe correta para escrever um comentário de várias linhas em JavaScript?",
  "options": ["// Comentário", "/ Comentário */", "<!-- Comentário -->", "# Comentário"],
  "answer": 1
},
{
  "question": "Qual é o método utilizado para ordenar os elementos de um array em JavaScript?",
  "options": ["sort()", "order()", "arrange()", "concat()"],
  "answer": 0
},
{
  "question": "Qual é a função utilizada para converter um número em uma string em JavaScript?",
  "options": ["parseInt()", "parseFloat()", "Number()", "toString()"],
  "answer": 3
},
{
  "question": "Qual é a forma correta de escrever uma estrutura de repetição 'while' em JavaScript?",
```

```

"options": ["while (condition) { // code }", "{ while condition // code }", "while { condition // code }", "while { condition: // code }"],
"answer": 0
},
{
"question": "Qual é o operador utilizado para comparar se dois valores são iguais em JavaScript?",
"options": ["=", "==", "===", "!="],
"answer": 2
},
{
"question": "Qual é a função utilizada para encontrar a posição de um elemento em um array em JavaScript?",
"options":

```

Aplicativo pra gerar parágrafos de um textarea onde será colocado o texto e ele separa as palavras, embaralha e mostra em formato json em outro textarea pra download

Aqui está um exemplo de aplicativo JavaScript que recebe um texto em um **textarea**, separa as palavras, embaralha e exibe o resultado em formato JSON em outro **textarea** para download:

```

<!DOCTYPE html>
<html>
<head>
  <title>Gerador de Parágrafos</title>
  <script>
    function generateParagraphs() {
      var inputText = document.getElementById('inputTextarea').value;
      var words = inputText.split(' ');
      shuffleArray(words);
      var outputText = {
        paragraphs: words
      };
      document.getElementById('outputTextarea').value = JSON.stringify(outputText, null, 2);
    }

    function shuffleArray(array) {
      for (var i = array.length - 1; i > 0; i--) {
        var j = Math.floor(Math.random() * (i + 1));
        var temp = array[i];
        array[i] = array[j];
        array[j] = temp;
      }
    }
  </script>
</head>
<body>
  <h1>Gerador de Parágrafos</h1>
  <label for="inputTextarea">Texto de Entrada:</label>
  <textarea id="inputTextarea" rows="5" cols="50"></textarea>
  <br>
  <button onclick="generateParagraphs()">Gerar Parágrafos</button>
  <br>
  <label for="outputTextarea">Resultado em JSON:</label>
  <textarea id="outputTextarea" rows="5" cols="50" readonly></textarea>
</body>
</html>

```

Neste exemplo, o aplicativo possui duas áreas de texto: uma para inserir o texto de entrada e outra para exibir o resultado em formato JSON. Ao clicar no botão "Gerar Parágrafos", a função `generateParagraphs()` é chamada. Essa função divide o texto em palavras, embaralha a ordem das palavras e cria um objeto JSON com a chave "paragraphs" contendo o array de palavras embaralhadas. O resultado é exibido no `textarea` de saída em formato JSON.

Lembre-se de salvar o código acima em um arquivo HTML e abrir no navegador para testar o aplicativo.

Aplicativo financeiro

Aqui está um exemplo básico de um aplicativo financeiro em JavaScript que permite calcular o saldo atual com base em receitas e despesas:

```
<!DOCTYPE html>
<html>
<head>
  <title>Aplicativo Financeiro</title>
  <script>
    var saldo = 0;

    function adicionarReceita() {
      var receita = parseFloat(document.getElementById('receitaInput').value);
      if (isNaN(receita)) {
        alert('Por favor, insira um valor válido para a receita.');
```

```
</body>
</html>
```

Neste exemplo, o aplicativo financeiro possui duas seções: "Receitas" e "Despesas". O usuário pode adicionar valores de receitas e despesas nos campos de entrada e clicar nos botões correspondentes para atualizar o saldo atual exibido na página.

O saldo é mantido na variável `saldo`, que é inicializada como 0. As funções `adicionarReceita()` e `adicionarDespesa()` são responsáveis por ler os valores inseridos nos campos de entrada, atualizar o saldo com base nas receitas ou despesas e chamar a função `atualizarSaldo()` para exibir o saldo atualizado na página.

Lembre-se de salvar o código acima em um arquivo HTML e abrir no navegador para testar o aplicativo financeiro básico. Note que este é apenas um exemplo simples e não abrange todas as funcionalidades e recursos avançados de um aplicativo financeiro completo.

Coloque mais funções nesse aplicativo financeiro

Aqui está uma versão aprimorada do aplicativo financeiro com algumas funções adicionais:

```
<!DOCTYPE html>
<html>
<head>
  <title>Aplicativo Financeiro</title>
<script>
  var saldo = 0;

  function adicionarReceita() {
    var receita = parseFloat(document.getElementById('receitaInput').value);
    if (isNaN(receita)) {
      alert('Por favor, insira um valor válido para a receita.');
```

```

function exibirMensagem() {
  var mensagem = 'Seu saldo atual é de R$ ' + saldo.toFixed(2) + '.';
  if (saldo < 0) {
    mensagem += ' Cuidado, você está com saldo negativo!';
  } else if (saldo > 0) {
    mensagem += ' Parabéns, você está com saldo positivo!';
  } else {
    mensagem += ' Seu saldo está zerado.';
  }
  alert(mensagem);
}
</script>
</head>
<body>
  <h1>Aplicativo Financeiro</h1>

  <h2>Receitas</h2>
  <input type="number" id="receitaInput" placeholder="Valor da Receita">
  <button onclick="adicionarReceita()">Adicionar Receita</button>

  <h2>Despesas</h2>
  <input type="number" id="despesaInput" placeholder="Valor da Despesa">
  <button onclick="adicionarDespesa()">Adicionar Despesa</button>

  <h2 id="saldo">Saldo Atual: R$ 0.00</h2>

  <button onclick="limparSaldo()">Limpar Saldo</button>
  <button onclick="exibirMensagem()">Exibir Mensagem</button>
</body>
</html>

```

Nesta versão aprimorada, foram adicionadas duas funções extras: `limparSaldo()` e `exibirMensagem()`.

A função `limparSaldo()` redefine o saldo para 0 e atualiza o valor exibido na página. Isso permite reiniciar o saldo para um novo cálculo.

A função `exibirMensagem()` exibe uma mensagem com base no saldo atual. Dependendo se o saldo é negativo, positivo ou zero, a mensagem exibida é diferente. A mensagem inclui o valor do saldo atual e uma observação adicional.

Lembre-se de salvar o código acima em um arquivo HTML e abrir no navegador para testar a versão aprimorada do aplicativo financeiro.

Dada uma url e um campo da página mostrar um alert toda vez que o campo mudar

Para detectar mudanças em um campo específico de uma página a partir de uma URL fornecida, você pode usar o JavaScript para monitorar o campo e verificar se houve alterações. Aqui está um exemplo de código que faz isso:

```

<!DOCTYPE html>
<html>
<head>
  <title>Monitorar Mudanças no Campo</title>
  <script>
    var campoAnterior = '';

    function monitorarCampo() {
      var url = document.getElementById('urlInput').value;
      var campo = document.getElementById('campoInput').value;

```

```

// Verificar se a URL está vazia
if (url === '') {
    alert('Por favor, insira uma URL válida.');
```

```

    return;
}

// Obter o conteúdo da URL usando uma requisição assíncrona (AJAX)
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState === 4 && this.status === 200) {
        var conteudo = this.responseText;


        // Verificar se o campo mudou
        if (campoAnterior !== conteudo) {
            alert('O campo mudou!');
            campoAnterior = conteudo;
        }
    }
};
xhttp.open('GET', url, true);
xhttp.send();
}
</script>
</head>
<body>
    <h1>Monitorar Mudanças no Campo</h1>

    <label for="urlInput">URL:</label>
    <input type="text" id="urlInput" placeholder="Insira a URL">
    <br>
    <label for="campoInput">Campo:</label>
    <input type="text" id="campoInput" placeholder="Insira o ID do campo">
    <br>
    <button onclick="monitorarCampo()">Monitorar Campo</button>
</body>
</html>

```

Neste exemplo, você precisa fornecer a URL da página que contém o campo que deseja monitorar e o ID desse campo. Ao clicar no botão "Monitorar Campo", o código faz uma requisição assíncrona (AJAX) para obter o conteúdo da URL especificada. Em seguida, ele verifica se o campo atual é diferente do campo anteriormente armazenado. Se houver uma diferença, um alerta é exibido indicando que o campo mudou.

Lembre-se de que o código acima pode ser restrito pelo mesmo domínio devido a limitações de segurança do navegador. Isso significa que você só poderá monitorar mudanças em campos de uma página se ela estiver no mesmo domínio que o arquivo HTML com o código JavaScript.

From ChatGPT, designed with  FancyGPT