Javascript Básico

JavaScript é uma **linguagem de programação** amplamente usada para criar conteúdo interativo e dinâmico em páginas web. Ela é uma linguagem de alto nível, interpretada e baseada em protótipos, que permite a criação de funcionalidades que vão desde animações simples até aplicações web completas.

Características principais:

1. **Interatividade em Páginas Web:**
    - JavaScript é usado para manipular o comportamento de elementos HTML e CSS, como responder a cliques de botão, validar formulários, ou criar animações.
2. **Lado do Cliente e Lado do Servidor:**
    - Originalmente projetado para execução no navegador (lado do cliente), ele também pode ser usado no lado do servidor, por exemplo, com o **Node.js**.
3. **Multiparadigma:**
    - Suporta programação orientada a objetos, funcional e procedural, oferecendo flexibilidade para diferentes estilos de codificação.
4. **Executado no Navegador:**
    - Funciona diretamente nos navegadores web (como Chrome, Firefox, Edge), sem necessidade de compilação.
5. **Dinâmico:**

- Permite alterações em tempo real na estrutura e no conteúdo da página sem a necessidade de recarregar o navegador (via **DOM Manipulation** e **AJAX**).

6. **Ecossistema Amplo:**

   - Possui uma vasta gama de bibliotecas e frameworks, como React, Angular, e Vue.js, que facilitam o desenvolvimento de aplicações robustas.

Exemplo simples:

Este script exibe um alerta quando o botão é clicado:

```
<!DOCTYPE html>
<html>
<head>
    <title>Exemplo JavaScript</title>
</head>
<body>
    <button onclick="mostrarAlerta()">Clique aqui</button>

    <script>
        function mostrarAlerta() {
            alert("Olá! Este é um exemplo de JavaScript.");
        }
    </script>
</body>
</html>
```

O que torna o JavaScript essencial?

- **Versatilidade:** Além de navegadores, pode ser usado para desenvolvimento de servidores, aplicativos móveis e até mesmo jogos.
- **Base da Web:** Juntamente com HTML e CSS, é um dos pilares fundamentais para o desenvolvimento de sites modernos.
- **Interatividade em Tempo Real:** É amplamente usado para aplicações interativas, como chats, painéis em tempo real e mapas.

Como incluir JavaScript em uma página web

Você pode incluir JavaScript em uma página web de três maneiras principais:

---

1. JavaScript embutido (Inline Script):

Escreva o código JavaScript diretamente no atributo `onclick`, `onmouseover`, etc., de elementos HTML.

<!DOCTYPE html>

<html>

<head>

  <title>Exemplo Inline</title>

</head>

<body>

  <button onclick="alert('Olá! Este é um exemplo de JavaScript embutido.')">Clique aqui</button>

</body>

</html>

---

2. JavaScript interno (Internal Script):

Inclua o código JavaScript dentro de uma tag `<script>` no arquivo HTML.

<!DOCTYPE html>

<html>

```
<head>
    <title>Exemplo Interno</title>
    <script>
        function mostrarMensagem() {
            alert("Olá! Este é um exemplo de JavaScript interno.");
        }
    </script>
</head>
<body>
    <button onclick="mostrarMensagem()">Clique aqui</button>
</body>
</html>
```

---

3. JavaScript externo (External Script):

Escreva o código JavaScript em um arquivo separado com extensão `.js` e vincule-o ao HTML usando a tag `<script>`.

Arquivo **script.js**:

```
function mostrarMensagem() {
    alert("Olá! Este é um exemplo de JavaScript externo.");
}
```

Arquivo **index.html**:

```
<!DOCTYPE html>

<html>

<head>

    <title>Exemplo Externo</title>

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

</head>

<body>

    <button onclick="mostrarMensagem()">Clique aqui</button>

</body>

</html>
```

Sintaxe básica do JavaScript

Aqui estão alguns conceitos fundamentais:

1. **Comentários:**

    - Linha única: `// Este é um comentário`

Multilinha:

```
/*

Este é um

comentário multilinha
```

*/

-

2. **Exemplo simples:**

```
// Declara uma variável
let nome = "João";

// Mostra uma mensagem no console
console.log("Olá, " + nome + "!");

// Exibe um alerta
alert("Bem-vindo, " + nome + "!");
```

3. **Regras de sintaxe:**
    - Cada instrução termina com `;` (opcional, mas recomendado).
    - JavaScript diferencia maiúsculas e minúsculas:
        - `nome` e `Nome` são variáveis diferentes.

Você pode experimentar esses exemplos no console do navegador (pressionando `F12` ou `Ctrl+Shift+I` e acessando a aba "Console") ou salvar em um arquivo HTML para testar no navegador. 🚀

Variáveis e Tipos de Dados em JavaScript

Variáveis são usadas para armazenar informações que podem ser reutilizadas e manipuladas no código. Em JavaScript, você pode declarar variáveis usando as palavras-chave `let`, `const` ou `var`.

Declarando Variáveis

1. **`let` (mais usado):**
    - Usado para declarar variáveis que podem ter seu valor alterado.

Exemplo:
```
let idade = 25;

idade = 30; // Agora, idade é 30

console.log(idade); // Exibe 30
```

    -

2. **`const` (para valores constantes):**
    - Usado para variáveis cujo valor **não pode ser alterado** depois de atribuído.

Exemplo:
```
const pi = 3.14;

console.log(pi); // Exibe 3.14
```

    -

3. **`var` (menos usado):**
    - Um método mais antigo para declarar variáveis. `let` e `const` são preferidos.

Exemplo:

```
var nome = "Ana";

console.log(nome);
```

-

Tipos de Dados

1. Número (`number`):

- Inclui inteiros e números de ponto flutuante.

Exemplo:

```
let idade = 25; // Número inteiro

let altura = 1.75; // Número decimal
```

-

2. Texto (`string`):

- Representa texto, entre aspas simples ou duplas.

Exemplo:

```
let nome = "Maria";

let saudacao = 'Olá, tudo bem?';
```

-

3. Booleano (`boolean`):

- Representa valores **verdadeiro** ou **falso**.

Exemplo:

```
let ativo = true;

let maiorDeIdade = false;
```

-

4. Indefinido (`undefined`):

- Uma variável declarada, mas que ainda não recebeu um valor.

Exemplo:

```
let cidade;

console.log(cidade); // undefined
```

-

5. Nulo (`null`):

- Representa a ausência intencional de um valor.

Exemplo:

```
let semValor = null;
```

-

6. Objeto (`object`):

- Um tipo de dado usado para armazenar coleções de pares chave-valor.

Exemplo:

```
let pessoa = {
    nome: "João",
    idade: 30,
    ativo: true
};

console.log(pessoa.nome); // Exibe "João"
```

-

7. Array (uma lista):

- Um objeto especial que armazena uma coleção ordenada de valores.

Exemplo:

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

console.log(frutas[1]); // Exibe "banana"
```

-

Exemplo Prático

```
// Declarando variáveis

let nome = "Lucas";

const idade = 20;

let trabalha = true;


// Mostrando valores no console

console.log("Nome: " + nome); // Nome: Lucas

console.log("Idade: " + idade); // Idade: 20

console.log("Trabalha? " + trabalha); // Trabalha? true
```

Operadores em JavaScript

Os operadores em JavaScript permitem realizar operações em valores ou variáveis. Eles são classificados em diferentes categorias, dependendo de sua funcionalidade:

---

1. Operadores Aritméticos

Usados para realizar cálculos matemáticos.

Operador	Descrição	Exemplo	Resultado
`+`	Adição	`5 + 2`	`7`
`-`	Subtração	`5 - 2`	`3`
`*`	Multiplicação	`5 * 2`	`10`
`/`	Divisão	`5 / 2`	`2.5`
`%`	Módulo (resto)	`5 % 2`	`1`
`**`	Exponenciação	`5 ** 2`	`25`

Exemplo:

let a = 10, b = 3;

console.log(a + b); // 13

console.log(a % b); // 1

---

2. Operadores de Atribuição

Usados para atribuir valores às variáveis.

Operador	Exemplo	Equivalente
`=`	`x = 10`	`x = 10`
`+=`	`x += 5`	`x = x + 5`
`-=`	`x -= 5`	`x = x - 5`
`*=`	`x *= 5`	`x = x * 5`
`/=`	`x /= 5`	`x = x / 5`
`%=`	`x %= 5`	`x = x % 5`

Exemplo:

let x = 10;

x += 5; // x agora é 15

console.log(x);

---

3. Operadores de Comparação

Usados para comparar dois valores. O resultado é sempre um **booleano** (true ou false).

Operador	Descrição	Exemplo	Resultado
`==`	Igual (valor)	`5 == '5'`	`true`
`===`	Estritamente igual (valor e tipo)	`5 === '5'`	`false`
`!=`	Diferente (valor)	`5 != 3`	`true`
`!==`	Estritamente diferente (valor e tipo)	`5 !== '5'`	`true`
`<`	Menor que	`3 < 5`	`true`
`>`	Maior que	`5 > 3`	`true`
`<=`	Menor ou igual a	`5 <= 5`	`true`
`>=`	Maior ou igual a	`5 >= 3`	`true`

Exemplo:

```
console.log(10 > 5);  // true

console.log(5 === "5");  // false
```

---

4. Operadores Lógicos

Usados para combinar condições.

Operador	Descrição	Exemplo	Resultado
`&&`	E (AND)	`true && false`	`false`
` ` `		` ` `	Ou (OR)
`!`	Não (NOT)	`!true`	`false`

Exemplo:

```
let a = 10, b = 5, c = 20;

console.log(a > b && c > a);  // true

console.log(a > b || c < a);  // true

console.log(!(a > b));        // false
```

---

5. Operadores Unários

Operam em apenas um operando.

Operador	Descrição	Exemplo	Resultado
`+`	Converte para número	`+"5"`	`5`
`-`	Negativo ou inverso	`-10`	`-10`
`++`	Incremento	`x++`	`x + 1`
`--`	Decremento	`x--`	`x - 1`

Exemplo:

let num = 5;

console.log(++num); // 6 (incrementa antes de usar)

console.log(num--); // 6 (usa antes de decrementar)

---

6. Operador Ternário

Uma forma compacta de escrever um `if/else`.

Sintaxe:

condição ? valorSeVerdadeiro : valorSeFalso;

Exemplo:

let idade = 18;

let status = (idade >= 18) ? "Adulto" : "Menor de idade";

console.log(status); // Adulto

Estruturas de Controle em JavaScript

As estruturas de controle permitem que o fluxo do programa seja desviado com base em condições ou repetido enquanto uma condição for verdadeira.

---

1. Estruturas Condicionais

As estruturas condicionais controlam o fluxo de execução com base em condições.

1.1. `if`, `else if`, `else`

Executa um bloco de código se a condição for verdadeira.

Sintaxe:

```
if (condição) {
    // Código se a condição for verdadeira
} else if (outraCondição) {
    // Código se outraCondição for verdadeira
} else {
    // Código se nenhuma condição for verdadeira
}
```

Exemplo:

```
let idade = 20;

if (idade < 18) {
    console.log("Menor de idade");
```

```
} else if (idade >= 18 && idade < 60) {
    console.log("Adulto");
} else {
    console.log("Idoso");
}
```

1.2. `switch`

Escolhe entre diferentes blocos de código com base no valor de uma variável ou expressão.

Sintaxe:

```
switch (expressão) {
    case valor1:
        // Código para valor1
        break;
    case valor2:
        // Código para valor2
        break;
    default:
        // Código se nenhum valor corresponder
}
```

Exemplo:

```
let dia = 3;



switch (dia) {
    case 1:
        console.log("Segunda-feira");
        break;
    case 2:
        console.log("Terça-feira");
        break;
    case 3:
        console.log("Quarta-feira");
        break;
    default:
        console.log("Dia inválido");
}
```

---

2. Estruturas de Repetição

Usadas para executar um bloco de código várias vezes.

2.1. `for`

Usado quando sabemos o número de iterações.

Sintaxe:

```
for (início; condição; incremento) {
    // Código a ser executado
}
```

Exemplo:

```
for (let i = 1; i <= 5; i++) {
    console.log("Contagem: " + i);
}
```

2.2. `while`

Executa o código enquanto a condição for verdadeira.

Sintaxe:

```
while (condição) {
    // Código a ser executado
}
```

Exemplo:

```
let contador = 1;
```

```
while (contador <= 5) {
    console.log("Número: " + contador);
    contador++;
}
```

2.3. `do...while`

Semelhante ao `while`, mas garante que o código seja executado pelo menos uma vez, independentemente da condição.

Sintaxe:

```
do {
    // Código a ser executado
} while (condição);
```

Exemplo:

```
let numero = 1;

do {
    console.log("Número: " + numero);
    numero++;
} while (numero <= 5);
```

2.4. `for...in`

Usado para iterar sobre as **propriedades de um objeto**.

Exemplo:

```
let pessoa = { nome: "João", idade: 30, cidade: "São Paulo" };


for (let chave in pessoa) {
    console.log(chave + ": " + pessoa[chave]);
}
```

2.5. `for...of`

Usado para iterar sobre valores de **arrays** ou objetos iteráveis.

Exemplo:

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


for (let fruta of frutas) {
    console.log(fruta);
}
```

---

3. Palavras-chave úteis

1. **`break`:**
    - Sai de um loop ou `switch` imediatamente.

Exemplo:

```
for (let i = 1; i <= 5; i++) {
  if (i === 3) break;
  console.log(i); // Exibe 1, 2
}
```

2. **`continue`:**
    - Pula para a próxima iteração do loop.

Exemplo:

```
for (let i = 1; i <= 5; i++) {
  if (i === 3) continue;
  console.log(i); // Exibe 1, 2, 4, 5
}
```

Exercício para prática

Problema: Crie um programa que peça para o usuário digitar um número e exiba a tabuada desse número de 1 a 10.

Funções em JavaScript

Funções são blocos de código reutilizáveis que executam uma tarefa ou calculam um valor. Elas ajudam a organizar e evitar repetição no código.

---

1. Como Criar e Usar Funções

1.1. Declaração de uma Função

Uma função pode ser definida usando a palavra-chave `function`.

Sintaxe:

```
function nomeDaFuncao(parametro1, parametro2, ...) {
    // Corpo da função
    return valor; // Opcional
}
```

Exemplo:

```
function saudacao(nome) {
    return "Olá, " + nome + "!";
}

console.log(saudacao("Maria")); // Exibe "Olá, Maria!"
```

1.2. Chamando uma Função

Para executar uma função, basta usar seu nome seguido de parênteses:

```
saudacao("Lucas");
```

---

2. Funções com e sem Parâmetros

Com parâmetros: Recebem dados para serem usados na execução.

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


console.log(soma(3, 4)); // Exibe 7
```

-

Sem parâmetros: Não recebem dados; usam valores internos.

```
 function mostrarMensagem() {
    console.log("Olá, mundo!");
}


mostrarMensagem(); // Exibe "Olá, mundo!"
```

-

---

3. Funções Anônimas

Uma **função anônima** é uma função sem nome, geralmente atribuída a uma variável ou usada como argumento.

Exemplo:

```
const saudacao = function(nome) {
    return "Olá, " + nome + "!";
};


console.log(saudacao("João")); // Exibe "Olá, João!"
```

---

4. Arrow Functions (Funções de Seta)

Introduzidas no ES6, as **arrow functions** têm uma sintaxe mais concisa e não vinculam o contexto `this`.

Sintaxe:

```
const nomeFuncao = (parametro1, parametro2, ...) => {
    // Corpo da função
    return valor; // Retorno explícito
};
```

- Se a função tiver apenas uma instrução de retorno, você pode omitir as chaves `{}` e o `return`.
- Se houver um único parâmetro, pode-se omitir os parênteses `()`.

Exemplos:

Função com Retorno Explícito:

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


console.log(soma(5, 3)); // Exibe 8
```

1.

Função com Retorno Implícito:

```
 const multiplica = (a, b) => a * b;
console.log(multiplica(4, 2)); // Exibe 8
```

2.

Função com um único parâmetro:

```
 const dobro = n => n * 2;
console.log(dobro(5)); // Exibe 10
```

3.

---

5. Diferença entre Funções Tradicionais e Arrow Functions

Característica	Função Tradicional	Arrow Function
Sintaxe	Mais longa	Mais curta
`this` vinculado	Depende do contexto de chamada	Vinculado ao escopo onde foi criada
Uso de `return`	Sempre necessário para retornar valores	Pode ser implícito se for uma única expressão

Exemplo Comparativo:

```
// Função Tradicional

function quadrado(num) {

    return num * num;

}



// Arrow Function

const quadradoArrow = num => num * num;



console.log(quadrado(4));       // 16

console.log(quadradoArrow(4)); // 16
```

Exercício para Praticar

Problema:

Crie uma função chamada `calculaMedia` que receba três números como

parâmetros e retorne a média desses números. Depois, teste a função com diferentes valores.

Manipulação do DOM com JavaScript

O **DOM (Document Object Model)** é a estrutura hierárquica que representa os elementos de uma página HTML. Usando JavaScript, podemos acessar, alterar e interagir com essa estrutura para criar páginas dinâmicas.

---

1. Seleção de Elementos

Para manipular o DOM, primeiro precisamos selecionar os elementos que desejamos alterar.

1.1. `getElementById`

Seleciona um elemento pelo seu `id`.

Exemplo:

const titulo = document.getElementById("titulo");

console.log(titulo); // Mostra o elemento com o id "titulo"

1.2. `getElementsByClassName`

Seleciona todos os elementos com uma classe específica. Retorna uma coleção de elementos.

Exemplo:

const itens = document.getElementsByClassName("item");

console.log(itens[0]); // Mostra o primeiro elemento com a classe "item"

1.3. `getElementsByTagName`

Seleciona todos os elementos com uma determinada tag (ex: `div`, `p`).

Exemplo:

const paragrafos = document.getElementsByTagName("p");

console.log(paragrafos.length); // Mostra quantos `<p>` existem na página

1.4. `querySelector` e `querySelectorAll`

Usam seletores CSS para buscar elementos.

- **`querySelector`** retorna o primeiro elemento que corresponde ao seletor.
- **`querySelectorAll`** retorna todos os elementos que correspondem ao seletor.

Exemplo:

const titulo = document.querySelector("#titulo"); // Seleciona pelo id

const itens = document.querySelectorAll(".item"); // Seleciona pela classe

console.log(itens[1]); // Mostra o segundo elemento com a classe "item"

---

2. Alteração de Conteúdo

2.1. Alterar texto com `innerText`

Modifica o texto visível de um elemento.

Exemplo:

```
const titulo = document.getElementById("titulo");

titulo.innerText = "Novo Título!";
```

2.2. Alterar HTML com `innerHTML`

Modifica o conteúdo HTML interno de um elemento.

Exemplo:

```
const container = document.getElementById("conteudo");

container.innerHTML = "<p>Texto novo com HTML!</p>";
```

2.3. Alterar atributos com `setAttribute`

Modifica ou adiciona um atributo ao elemento.

Exemplo:

```
const link = document.querySelector("a");

link.setAttribute("href", "https://www.example.com");

link.innerText = "Ir para o site";
```

---

3. Alteração de Estilos

3.1. Alterar estilos com `style`

Usa a propriedade `style` para definir CSS diretamente no elemento.

Exemplo:

```
const titulo = document.getElementById("titulo");

titulo.style.color = "blue";

titulo.style.fontSize = "24px";
```

3.2. Adicionar ou remover classes com `classList`

Manipula as classes CSS de um elemento.

Método	Descrição
`add("classe")`	Adiciona uma classe
`remove("classe")`	Remove uma classe
`toggle("classe")`	Adiciona se não existir, remove se existir
`contains("classe")`	Verifica se o elemento possui uma classe

Exemplo:

```
const titulo = document.getElementById("titulo");

titulo.classList.add("destaque");

titulo.classList.remove("oculto");
```

---

4. Criação e Remoção de Elementos

4.1. Criar elementos com `createElement`

Cria um novo elemento HTML.

Exemplo:

```
const novoParagrafo = document.createElement("p");

novoParagrafo.innerText = "Este é um parágrafo criado dinamicamente.";

document.body.appendChild(novoParagrafo); // Adiciona ao final do body
```

4.2. Remover elementos com `remove`

Remove o elemento do DOM.

Exemplo:

```
const titulo = document.getElementById("titulo");

titulo.remove(); // Remove o elemento
```

4.3. Inserir elementos com `appendChild` e `insertBefore`

- `appendChild`: Insere um elemento no final de outro.
- `insertBefore`: Insere um elemento antes de outro.

Exemplo:

```
const lista = document.getElementById("lista");
```

```
const novoItem = document.createElement("li");

novoItem.innerText = "Novo Item";

lista.appendChild(novoItem); // Adiciona ao final da lista
```

---

5. Eventos

Os eventos permitem que você execute ações quando algo ocorre (ex: clique, teclado, etc.).

5.1. Adicionar eventos com `addEventListener`

Escuta um evento e executa uma função.

Exemplo:

```
const botao = document.getElementById("botao");

botao.addEventListener("click", () => {
    alert("Você clicou no botão!");
});
```

5.2. Eventos comuns

- `click`: Clique do usuário.
- `mouseover`: Mouse sobre o elemento.
- `mouseout`: Mouse sai do elemento.
- `keydown`: Tecla pressionada.
- `load`: Página ou recurso carregado.

Exemplo Prático

Crie um botão que muda o texto de um parágrafo quando clicado.

HTML:

```
<p id="texto">Texto original</p>

<button id="botao">Clique aqui</button>
```

JavaScript:

```
const botao = document.getElementById("botao");

const texto = document.getElementById("texto");


botao.addEventListener("click", () => {

    texto.innerText = "Texto alterado!";

});
```

Eventos em JavaScript

Os **eventos** em JavaScript permitem que você responda a interações do usuário, como cliques de botão, pressionamento de teclas ou movimentação do mouse. Eles são essenciais para criar páginas interativas.

---

1. Adicionando Eventos

1.1. Atributo HTML `onclick` (não recomendado)

Você pode adicionar eventos diretamente no HTML, mas essa prática não é ideal, pois dificulta a manutenção do código.

Exemplo:

<button onclick="alert('Botão clicado!')">Clique aqui</button>

1.2. Usando `addEventListener` (recomendado)

A maneira moderna e recomendada é usar o método `addEventListener` para associar eventos a elementos no JavaScript.

Exemplo:

```
const botao = document.getElementById("botao");

botao.addEventListener("click", () => {
    alert("Botão clicado!");
});
```

---

2. Eventos Comuns

2.1. Evento de Clique (`click`)

Executado quando o usuário clica em um elemento.

Exemplo:

```
document.querySelector("#meuBotao").addEventListener("click", () => {
    console.log("O botão foi clicado!");
});
```

2.2. Evento de Teclado

Captura as teclas pressionadas pelo usuário.

- `keydown`: Disparado quando uma tecla é pressionada.
- `keyup`: Disparado quando uma tecla é solta.

Exemplo:

```
document.addEventListener("keydown", (event) => {
    console.log("Tecla pressionada: " + event.key);
});
```

---

3. Manipulando o Objeto de Evento

O objeto de evento, passado automaticamente para o manipulador, fornece informações sobre o evento (ex.: o elemento que disparou o evento, tecla pressionada, etc.).

Exemplo:

```
document.querySelector("#botao").addEventListener("click", (event) => {
    console.log("Elemento clicado:", event.target); // Mostra o elemento que foi clicado
});
```

---

4. Outros Eventos Úteis

4.1. `mouseover` e `mouseout`

Detectam quando o mouse entra ou sai de um elemento.

Exemplo:

```
const caixa = document.getElementById("caixa");

caixa.addEventListener("mouseover", () => {
    caixa.style.backgroundColor = "lightblue";
});

caixa.addEventListener("mouseout", () => {
    caixa.style.backgroundColor = "white";
});
```

4.2. `change`

Disparado quando o valor de um elemento de formulário muda.

Exemplo:

```
document.getElementById("meuInput").addEventListener("change", (event) => {
    console.log("Valor alterado para:", event.target.value);
});
```

4.3. `submit`

Usado para capturar o envio de formulários.

Exemplo:

```
document.querySelector("form").addEventListener("submit", (event) => {
    event.preventDefault(); // Evita o envio padrão do formulário
    console.log("Formulário enviado!");
});
```

---

5. Removendo Eventos

Você pode remover um evento com o método `removeEventListener`. Para isso, o manipulador precisa ser declarado como uma função nomeada.

Exemplo:

```
const clicar = () => console.log("Clique detectado!");
const botao = document.getElementById("botao");
```

```
botao.addEventListener("click", clicar);

botao.removeEventListener("click", clicar); // Remove o evento
```

Exemplo Prático

Crie um campo de texto que mostre o que o usuário digita em tempo real.

HTML:

```
<input type="text" id="campo" placeholder="Digite algo...">

<p id="saida"></p>
```

JavaScript:

```
const campo = document.getElementById("campo");

const saida = document.getElementById("saida");


campo.addEventListener("input", (event) => {

    saida.innerText = "Você digitou: " + event.target.value;

});
```
