E-BOOK GRATUITO

JAVASCRIPT DO ZERO

Um guia direto para entender lógica, criar interações no navegador e começar suas primeiras aplicações web.

`<html>`	`const ideia = "código";`	`{ estilo: true }`
ESTRUTURA	**COMPORTAMENTO**	**VISUAL**

PROFESSOR MARCOS
Desvendando o Código

BOAS-VINDAS

01. Programação começa pelo raciocínio

Antes de frameworks, APIs ou bancos de dados, existe uma habilidade que sustenta todas as outras: transformar um problema em passos que o computador consiga executar.

Este e-book foi criado para quem está começando e sente que os códigos parecem um conjunto de palavras difíceis. O objetivo não é decorar comandos. É compreender a lógica por trás deles e perceber como cada parte se conecta.

O QUE VOCÊ VAI CONSEGUIR	Ler pequenos códigos, entender decisões e repetições, manipular elementos de uma página e visualizar o caminho até um projeto real.

A melhor forma de aprender programação é praticar em etapas. Leia os exemplos, digite por conta própria, altere valores e observe os resultados. O erro faz parte do processo: ele mostra exatamente onde seu raciocínio precisa ser ajustado.

“Você não precisa saber tudo para começar. Precisa começar para construir uma base.”

ROTEIRO

02. Como aproveitar este guia

Use o material como uma pequena trilha. Cada assunto prepara o próximo e culmina em aplicações que reagem às ações do usuário.

- Leia o conceito: procure compreender o problema antes do código.
- Digite o exemplo: copiar e colar é rápido, mas digitar ajuda a perceber a estrutura.
- Faça uma alteração: troque números, mensagens, cores ou condições.
- Teste erros: remova uma chave ou altere um nome para aprender a interpretar o console.
- Registre dúvidas: anotar a pergunta ajuda a organizar o raciocínio.

TEMPO SUGERIDO	Estude de 20 a 40 minutos por sessão. Parar, testar e voltar costuma ser mais eficiente que assistir muitas aulas sem praticar.

Não se preocupe se um exemplo parecer simples. Variáveis, condicionais, funções e arrays reaparecem em aplicações maiores. O que muda é a quantidade de dados e regras envolvidas.

VISÃO GERAL

03. O caminho do JavaScript básico

A trilha completa foi organizada para sair do raciocínio inicial e chegar à construção de uma aplicação visual.

ETAPA	CONTEÚDO
1	Lógica, algoritmos e formas de representar uma solução
2	HTML, CSS, editor e estrutura de uma página
3	Variáveis, tipos, entrada, saída e conversão
4	Operadores e estruturas condicionais
5	Fluxo das aplicações web e organização do código
6	DOM, elementos dinâmicos e manipulação do HTML
7	Eventos e interatividade
8	Formulários e tratamento de dados
9	Arrays e estruturas de repetição
10	Funções e reutilização
11	Projeto final: Dev Store

FUNDAMENTOS

04. O que é um algoritmo?

Um algoritmo é uma sequência organizada de passos para resolver um problema. Antes de pensar na linguagem, precisamos saber o que deve acontecer.

Imagine o processo de calcular a média de duas notas. Primeiro recebemos os valores, depois realizamos o cálculo e, por fim, mostramos o resultado. Esse fluxo pode ser representado como entrada, processamento e saída.

DESCRIÇÃO NARRATIVA

```
Entrada: nota1 e nota2
Processamento: (nota1 + nota2) / 2
Saída: média calculada
```

PENSE ANTES DE CODIFICAR	Qual é o problema? Quais dados entram? Quais regras transformam esses dados? Qual resultado precisa ser apresentado?

Quando o problema está claro, escrever o código fica muito mais simples. Essa forma de decompor situações também ajuda a identificar erros e testar cada etapa separadamente.

WEB

05. HTML, CSS e JavaScript

As três tecnologias trabalham juntas, mas cada uma possui uma responsabilidade diferente dentro da página.

HTML - ESTRUTURA

```
<h1>Minha página</h1>
<button id="botao">Clique aqui</button>
```

CSS - APARÊNCIA

```
button {
  background: #ffd400;
  color: #111;
}
```

JAVASCRIPT - COMPORTAMENTO

```
const botao = document.getElementById("botao");
botao.addEventListener("click", () => {
  alert("Funcionou!");
});
```

Separar essas responsabilidades melhora a organização. O HTML descreve os elementos, o CSS define como eles aparecem e o JavaScript determina como respondem às ações.

PRIMEIROS CÓDIGOS

06. Variáveis e tipos de dados

Variáveis guardam valores para que eles possam ser utilizados e alterados durante a execução do programa.

TIPOS BÁSICOS

```
const nome = "Marcos";       // string
let idade = 40;              // number
const estudando = true;      // boolean
let resultado;               // undefined
```

const é utilizada quando a variável não receberá uma nova atribuição. let permite trocar o valor posteriormente. O nome escolhido deve explicar o que aquela informação representa.

NOMES CLAROS	Prefira quantidadeProdutos em vez de x. Um nome descritivo economiza tempo quando você precisar revisar ou corrigir o código.

O JavaScript identifica o tipo pelo valor armazenado. Mesmo assim, compreender os tipos é essencial, porque números e textos se comportam de maneiras diferentes.

ENTRADA DE DADOS

07. Concatenação e conversão

Valores digitados em campos ou capturados pelo prompt normalmente chegam como texto. Isso explica um dos erros mais comuns de quem está começando.

O PROBLEMA

```
const valor1 = prompt("Primeiro número:");
const valor2 = prompt("Segundo número:");

console.log(valor1 + valor2); // 20 e 30 viram 2030
```

A CONVERSÃO

```
const numero1 = Number(valor1);
const numero2 = Number(valor2);

console.log(numero1 + numero2); // 50
```

REGRA PRÁTICA	Antes de calcular, confirme se os dados realmente são números. A interface pode mostrar dígitos, mas o JavaScript ainda pode tratá-los como texto.

A concatenação junta textos. A soma realiza uma operação matemática. O operador é o mesmo, mas o resultado depende dos tipos envolvidos.

DECISÕES

08. Operadores e condicionais

Uma aplicação precisa tomar decisões. Para isso, comparamos valores e executamos blocos diferentes conforme o resultado.

IF E ELSE

```
const idade = 18;

if (idade >= 18) {
  console.log("Entrada permitida");
} else {
  console.log("Entrada não permitida");
}
```

A expressão idade >= 18 resulta em verdadeiro ou falso. Quando é verdadeira, o primeiro bloco executa; caso contrário, o programa segue para o else.

OPERADORES ÚTEIS	Comparação: ===, !==, >, <, >= e <=. Lógicos: && para exigir várias condições e || para aceitar alternativas.

Uma boa condição precisa representar a regra do problema. Não basta o código funcionar: ele deve decidir exatamente o que foi solicitado.

PRÁTICA

09. Validação antes de continuar

Validar significa impedir que dados inadequados avancem pelo programa. Uma verificação simples evita resultados incorretos e melhora a experiência do usuário.

CAMPO OBRIGATÓRIO

```
function cadastrar() {
  const nome = inputNome.value.trim();

  if (nome === "") {
    mensagem.textContent = "Digite seu nome";
    return;
  }

  mensagem.textContent = `Bem-vindo, ${nome}!`;
}
```

O método trim() remove espaços das extremidades. Assim, um campo preenchido somente com espaços continua sendo considerado vazio.

POR QUE RETURN?	Ao encontrar um dado inválido, return encerra a função naquele ponto. O restante do cadastro não é executado.

Outras validações básicas podem utilizar length para tamanho mínimo e includes() para verificar a presença de um caractere.

REPETIÇÕES

10. Quando usar um laço?

Repetir comandos manualmente funciona com poucos dados, mas não escala. Laços executam a mesma regra para diferentes valores.

LAÇO FOR

```
for (let contador = 1; contador <= 5; contador++) {
  console.log(contador);
}
```

O for possui três partes: valor inicial, condição de continuidade e atualização do contador. Se a condição nunca se tornar falsa, teremos um loop infinito.

LAÇO WHILE

```
let numero = 1;

while (numero <= 3) {
  console.log(numero);
  numero++;
}
```

Use for quando o controle da repetição estiver bem definido. Use while quando a repetição depender de uma condição que pode variar durante a execução.

COLEÇÕES

11. Arrays organizam listas

Um array reúne vários valores em uma única variável. Cada item possui uma posição, chamada índice, e a contagem começa em zero.

CRIANDO UM ARRAY

```
const linguagens = ["JavaScript", "Python", "Java"];

console.log(linguagens[0]); // JavaScript
console.log(linguagens.length); // 3
```

Arrays podem crescer e diminuir. O método push() adiciona ao final, unshift() adiciona no início, pop() remove o último e shift() remove o primeiro.

ALTERANDO A LISTA

```
linguagens.push("C#");
linguagens.splice(1, 1);

console.log(linguagens);
```

ATENÇÃO AO ÍNDICE O primeiro elemento está no índice 0. Em um array com três itens, o último índice é 2.

ARRAYS

12. Percorrendo todos os itens

Quando os dados estão em um array, podemos executar uma ação para cada elemento sem repetir o código manualmente.

FOR COM ÍNDICE

```
const nomes = ["Ana", "Carlos", "Luiza"];

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

FOREACH

```
nomes.forEach((nome) => {
  console.log(nome);
});
```

O forEach recebe uma função que será executada para cada item. Essa ideia é muito utilizada para transformar listas de dados em elementos visuais, como produtos, usuários ou tarefas.

CONEXÃO COM PROJETOS	A Dev Store utiliza forEach para percorrer um array de produtos e criar automaticamente um card para cada objeto.

REUTILIZAÇÃO

13. Funções dividem o problema

Funções agrupam instruções que realizam uma tarefa. Elas ajudam a separar responsabilidades e evitam repetir regras.

PARÂMETROS E RETORNO

```
function calcularMedia(nota1, nota2) {
  const media = (nota1 + nota2) / 2;
  return media;
}

const resultado = calcularMedia(8, 7);
console.log(resultado);
```

Parâmetros são os nomes utilizados na definição. Argumentos são os valores enviados na chamada. O return devolve o resultado para que ele possa ser reutilizado.

UMA RESPONSABILIDADE	Funções menores são mais fáceis de entender, testar e corrigir. Em vez de uma função gigante, divida o fluxo em etapas claras.

Uma calculadora de IMC, por exemplo, pode separar captura dos campos, cálculo, classificação e exibição do resultado.

NAVEGADOR

14. O que é DOM?

O DOM representa os elementos do documento HTML como objetos que podem ser encontrados e modificados pelo JavaScript.

HTML

```
<h2 id="titulo">Olá</h2>
<button id="alterar">Alterar</button>
```

JAVASCRIPT

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

botao.addEventListener("click", () => {
  titulo.textContent = "JavaScript funcionando!";
});
```

O método getElementById() procura o elemento pelo atributo id. Depois de selecioná-lo, podemos alterar texto, estilos, classes e outros conteúdos.

VALUE OU TEXTCONTENT?	Use value para ler campos de formulário. Use textContent para ler ou alterar o texto de elementos como h1, p e div.

INTERATIVIDADE

15. Eventos conectam ações ao código

Um evento ocorre quando o usuário clica, digita, envia um formulário ou realiza outra ação reconhecida pelo navegador.

EVENTO DE CLIQUE

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

botao.addEventListener("click", function () {
  console.log("O botão foi clicado");
});
```

O addEventListener registra uma função para responder ao evento. Essa função pode ser convencional, anônima ou uma arrow function.

EVENTO DE TECLADO

```
input.addEventListener("keypress", (event) => {
  if (event.key === "Enter") {
    adicionarItem();
  }
});
```

Eventos transformam uma página estática em uma aplicação interativa. A regra fica no JavaScript; o evento apenas define quando ela deve ser executada.

FORMULÁRIOS

16. Tratando o envio dos dados

Ao enviar um formulário, o navegador tenta recarregar a página. Em aplicações JavaScript, normalmente interrompemos esse comportamento para tratar os dados.

SUBMIT E PREVENTDEFAULT

```
formulario.addEventListener("submit", (event) => {
  event.preventDefault();

  const email = inputEmail.value.trim();

  if (!email.includes("@")) {
    mensagem.textContent = "E-mail inválido";
    return;
  }

  mensagem.textContent = "Dados enviados";
});
```

Uma validação no navegador melhora a experiência, mas aplicações reais também precisam validar no servidor. Cada camada protege o sistema em um ponto diferente.

NÃO CONFIE APENAS NO HTML	Atributos como required ajudam, mas regras importantes também devem existir no JavaScript e, posteriormente, no back-end.

DOM DINÂMICO

17. Criando elementos pela programação

O JavaScript pode criar itens que não estavam escritos originalmente no HTML. Essa técnica é a base de listas, tabelas, cards e catálogos.

CRIANDO UM ITEM

```
const item = document.createElement("li");
item.textContent = "Estudar JavaScript";

lista.appendChild(item);
```

ADICIONANDO UMA AÇÃO

```
const botaoRemover = document.createElement("button");
botaoRemover.textContent = "Remover";

botaoRemover.addEventListener("click", () => {
  item.remove();
});

item.appendChild(botaoRemover);
```

O elemento é criado, configurado e só então anexado à página. Para remover, podemos utilizar remove() ou localizar o elemento pai e chamar removeChild().

ORGANIZAÇÃO

18. Planeje antes de construir

Projetos maiores ficam mais simples quando dividimos a implementação em blocos pequenos e verificáveis.

Uma possível organização para uma aplicação com produtos é:

- Definir ou receber a lista de produtos.
- Capturar os elementos necessários do HTML.
- Criar funções auxiliares, como a formatação do preço.
- Percorrer os dados e montar os cards.
- Adicionar eventos aos botões.
- Atualizar o carrinho e mostrar notificações.
- Testar a responsividade e os casos de erro.

CHECKLIST DE CÓDIGO	Execute e teste cada bloco antes de seguir. Quando algo quebrar, você saberá qual foi a última alteração realizada.

Comentários podem funcionar como um roteiro temporário. Depois que o código estiver estável, mantenha apenas os que realmente ajudam a explicar decisões importantes.

PROJETO FINAL

19. Como funciona a Dev Store

A Dev Store é uma vitrine de produtos construída com HTML, CSS e JavaScript. O catálogo nasce de um array de objetos e é renderizado automaticamente.

ARRAY DE OBJETOS

```
const produtos = [
  {
    id: 1,
    nome: "Mouse Gamer RGB",
    categoria: "Periféricos",
    preco: 129.90,
    desconto: 28,
    icone: "□□"
  }
];
```

Cada objeto representa um produto e agrupa suas propriedades. Se novos objetos forem adicionados ao array, o mesmo algoritmo consegue gerar novos cards sem reescrever toda a estrutura.

VALOR DO PROJETO	Aqui os fundamentos deixam de aparecer isolados: arrays, objetos, repetição, funções, DOM, eventos e interface passam a trabalhar juntos.

PROJETO FINAL

20. Renderizando os cards

O forEach percorre os produtos. Para cada item, o JavaScript cria um article, aplica a classe visual e insere os dados usando template literal.

ESTRUTURA SIMPLIFICADA

```
produtos.forEach((produto) => {
  const card = document.createElement("article");
  card.classList.add("card-produto");

  card.innerHTML = `
    <h3>${produto.nome}</h3>
    <p>${produto.categoria}</p>
    <strong>${formatarPreco(produto.preco)}</strong>
    <button class="botao-comprar">Adicionar</button>
  `;

  listaProdutos.appendChild(card);
});
```

O template literal permite combinar texto, HTML e valores do JavaScript. A interpolação é feita com ${ }. Como o navegador não corrige automaticamente essa estrutura, aspas, crases e fechamentos exigem atenção.

UMA ESTRUTURA, MUITOS CARDS A quantidade de produtos pode mudar, mas a lógica de renderização permanece a mesma.

PROJETO FINAL

21. Carrinho e notificações

Cada botão criado no card recebe um evento. Ao clicar, a aplicação atualiza o contador, altera temporariamente o botão e mostra uma mensagem.

FUNÇÃO DO CARRINHO

```
function adicionarAoCarrinho(produto, botao) {
  quantidadeItens++;
  quantidadeCarrinho.innerText = quantidadeItens;

  botao.innerText = "Adicionado ✓";
  botao.classList.add("adicionado");

  setTimeout(() => {
    botao.innerText = "Adicionar ao carrinho";
    botao.classList.remove("adicionado");
  }, 1200);

  mostrarNotificacao(produto.nome);
}
```

setTimeout agenda uma ação futura. classList adiciona ou remove estados visuais sem misturar toda a estilização dentro do JavaScript.

EXPERIÊNCIA DO USUÁRIO	Uma boa interface confirma a ação realizada. O contador, o texto do botão e a notificação mostram que o clique foi processado.

DESAFIO

22. Personalize sua própria loja

Depois de compreender o projeto, o melhor exercício é modificá-lo até que deixe de parecer apenas uma cópia.

- Troque o nome, o texto principal e as cores da loja.
- Crie pelo menos quatro novos produtos.
- Adicione uma propriedade chamada estoque.
- Mostre uma mensagem quando o estoque for igual a zero.
- Crie categorias diferentes e altere os ícones.
- Inclua um botão para remover um item do carrinho.
- Teste a aplicação em tamanhos diferentes de tela.

DESAFIO EXTRA	Crie um campo de busca e utilize filter() para apresentar apenas os produtos cujo nome contém o texto digitado.

Personalizar obriga você a interpretar o código e tomar decisões. É nesse momento que o projeto começa a se transformar em conhecimento próprio.

DEPURAÇÃO

23. Erros fazem parte do desenvolvimento

Quando algo não funcionar, evite alterar várias partes ao mesmo tempo. Leia a mensagem do console e investigue de forma organizada.

- ReferenceError: normalmente indica nome inexistente ou variável fora do escopo.
- SyntaxError: procure chaves, parênteses, aspas ou crases sem fechamento.
- Elemento null: confirme o id e se o script executa depois do HTML.
- NaN: verifique se o valor foi convertido corretamente para número.
- Evento não dispara: confirme se o elemento foi selecionado e se o nome do evento está correto.

PERGUNTAS PARA O CONSOLE

```
console.log(valor);
console.log(typeof valor);
console.log(elemento);
```

MÉTODO	Reproduza o problema, localize o último ponto que funcionou, confira os valores e faça uma alteração por vez.

REVISÃO

24. O que você já consegue reconhecer

Chegar até aqui significa que os principais blocos de uma aplicação JavaScript já não são completamente desconhecidos.

- Transformar um problema em entrada, processamento e saída.
- Entender o papel de HTML, CSS e JavaScript.
- Criar variáveis e diferenciar textos, números e booleanos.
- Converter valores antes de realizar cálculos.
- Construir decisões com if e else.
- Repetir ações com for, while e forEach.
- Organizar listas com arrays.
- Criar funções com parâmetros e retorno.
- Selecionar, alterar, criar e remover elementos do DOM.
- Responder a cliques, teclas e envio de formulários.

Esses fundamentos aparecem novamente no front-end, no back-end e em frameworks. Quanto mais firme estiver essa base, mais natural será aprender tecnologias novas.

PRÓXIMO PASSO

25. Do guia para a prática completa

Este e-book apresentou uma visão do caminho. O curso organiza a aprendizagem em vídeos curtos e progressivos, com explicações práticas e um projeto final.

NO CURSO	O QUE VOCÊ RECEBE
11 módulos	Uma sequência organizada dos fundamentos até a Dev Store.
142 capítulos	Aulas objetivas para estudar em períodos curtos.
Projetos e códigos	Materiais de apoio para acompanhar as aplicações maiores.
Projeto final	Construção de uma vitrine dinâmica com carrinho e notificações.

ACOMPANHE O LANÇAMENTO	Acesse desvendandoocodigo.com.br para encontrar conteúdos, projetos, aulas e as novidades do curso completo.

Continue praticando. Seu próximo código começa com um problema pequeno e a decisão de tentar resolvê-lo.