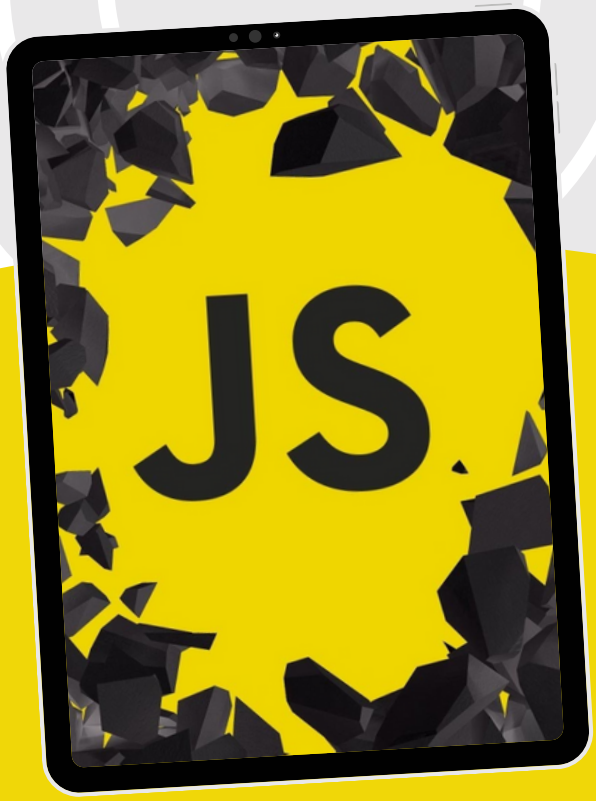
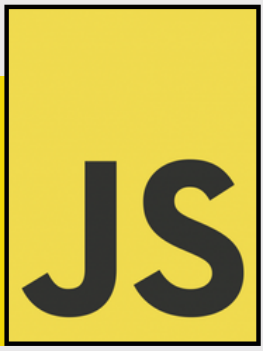
E-BOOK
INTRODUÇÃO
JAVA SCRIPT

Desvende os Segredos da Linguagem de Programação Mais Poderosa da Web



CAPÍTULO 1: CONCEITOS BÁSICOS



VARIÁVEIS E TIPOS DE DADOS

Java Script é uma linguagem de programação que lida com diferentes tipos de dados. Os tipos de dados mais comuns são:

- **Number**: Representa valores numéricos. **Exemplo:**

```
let idade = 30;
```

- **String**: Representa texto. É sempre colocado entre aspas simples (') ou duplas ("). **Exemplo:**

```
let nome = "João";
```

- **Boolean**: Representa verdadeiro ou falso. **Exemplo:**

```
let estaChovendo = false;
```

- **Null e Undefined**: Representam a ausência de valor. **null** é uma atribuição intencional de ausência de valor, enquanto **undefined** significa que a variável não foi inicializada. **Exemplo:**

```
let valorNulo = null;
let valorIndefinido;
```

VARIÁVEIS E TIPOS DE DADOS

- **Array:** Armazena uma coleção de elementos. Exemplo:

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

- ***Object**: Representa um conjunto de dados ou propriedades. **Exemplo:***

```
let pessoa = {
nome: "Maria",
idade: 25,
cidade: "São Paulo"};
```

OPERADORES

Os operadores são símbolos que realizam operações em variáveis e valores. Alguns exemplos comuns são:

- **Aritméticos:** + (adição), - (subtração), * (multiplicação), / (divisão), % (módulo). **Exemplo:**

```
let soma = 10 + 5; // soma = 15
```

- **Comparação:** == (igual a), != (diferente de), > (maior que), < (menor que), >= (maior ou igual a), <= (menor ou igual a). **Exemplo:**

```
let a = 10;
let b = 5;
let maior = a > b; // maior = true
```

OPERADORES

- **Lógicos:** && (E lógico), || (OU lógico), ! (NÃO lógico). **Exemplo:**

```
let chuva = true;
let vento = false;
let ficarEmCasa = chuva && !vento;
// ficarEmCasa = true
```

CONTROLE DE FLUXO

O controle de fluxo permite que o programa tome decisões baseadas em condições. Um exemplo é a **estrutura if-else**:

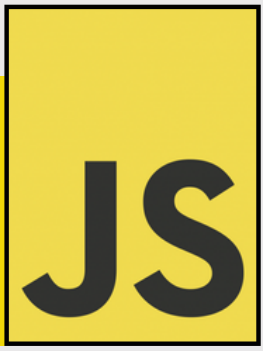
```
let idade = 18;
if (idade >= 18) {
 console.log("Você é maior de idade.");
} else {
 console.log("Você é menor de idade.");}
```

Outra estrutura importante é o **loop for**:

```
for (let i = 0; i < 5; i++) {
console.log("Número: " + i);}
```

Neste exemplo, o loop irá imprimir os números de 0 a 4. Estes são os conceitos básicos para começar a programar em JavaScript. No próximo capítulo, iremos explorar funções e escopo de variáveis.

CAPÍTULO 2: FUNÇÕES



DEFINIÇÃO DE FUNÇÕES

Funções são blocos de código reutilizáveis que podem ser chamados para executar uma tarefa específica. Elas ajudam a organizar e modularizar o código.

```
function saudacao(nome) {
 console.log("Olá, " + nome + "!"); }
saudacao("Maria"); // Saída: Olá, Maria!
```

ESCOPO DE VARIÁVEIS

O escopo de uma variável determina onde ela pode ser acessada no código. Existem dois tipos principais de escopo:

- **Escopo Global:** Variáveis globais são declaradas fora de qualquer função e podem ser acessadas de qualquer lugar no código.

```
let global = "Isso é global";
function funcao() {
    console.log(global);  // Saída: Isso é
global }
```

ESCOPO DE VARIÁVEIS

- *Escopo Local: Variáveis declaradas dentro de uma função só podem ser acessadas dentro dessa função.*

```
function funcao() {
 let local = "Isso é local";
 console.log(local); // Saída: Isso é local
}
console.log(local); // Erro: local is not
defined
```

RETORNO DE VALORES

- *Funções podem retornar valores usando a palavra-chave return.*

```
function soma(a, b) {
 return a + b;
}
let resultado = soma(3, 5); // resultado = 8
```

FUNÇÕES DE ORDEM SUPERIOR

Em JavaScript, funções são tratadas como cidadãos de primeira classe, o que significa que podem ser passadas como argumentos para outras funções e até mesmo retornadas como valores.

```
function operacaoMatematica(a, b, operacao) {
 return operacao(a, b);
}
function soma(a, b) {
 return a + b;
}
function subtracao(a, b) {
 return a - b;
}
let resultadoSoma = operacaoMatematica(5, 3, soma); // resultadoSoma = 8
let resultadoSubtracao = operacaoMatematica(5, 3, subtracao); // resultadoSubtracao = 2
```

No próximo capítulo, vamos explorar objetos e classes em JavaScript.

CAPÍTULO 3: OBJETOS E CLASSES

INTRODUÇÃO A OBJETOS

Em JavaScript, objetos são coleções de pares chave-valor. Eles são fundamentais para organizar e manipular dados.

```
let pessoa = {
 nome: "Maria",
 idade: 25,
 cidade: "São Paulo"
};
```

PROPRIEDADES E MÉTODOS

- **Propriedades:** São os dados associados a um objeto.

```
console.log(pessoa.nome); // Saída: Maria
```

- ***Métodos**: São funções associadas a um objeto.*

```
let pessoa = {
 nome: "Maria",
 saudacao: function() {
  console.log("Olá, eu sou "
+ this.nome);  }};
```

```
pessoa.saudacao(); //
Saída: Olá, eu sou Maria
```

CLASSES E CONSTRUTORES

Classes são modelos para criar objetos. Os construtores são funções que são chamadas quando uma nova instância de uma classe é criada.

Exemplo:

```
class Pessoa {
 constructor(nome, idade, cidade) {
  this.nome = nome;
  this.idade = idade;
  this.cidade = cidade;
 }

 saudacao() {
  console.log("Olá, eu sou " + this.nome);
 }
}
let maria = new Pessoa("Maria", 25, "São Paulo");
maria.saudacao(); // Saída: Olá, eu sou Maria
```

HERANÇA E PROTÓTIPOS

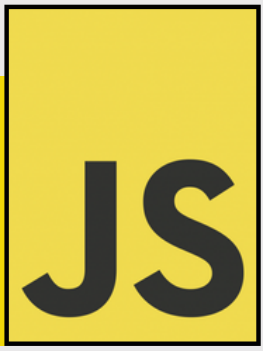
A herança permite que uma classe herde propriedades e métodos de outra classe.

Exemplo:

```
class Animal {
 comer() {
  console.log("Estou comendo");
 }
}

class Cachorro extends Animal {
 latir() {
  console.log("Au au!");
 }
}

let meuCachorro = new Cachorro();
meuCachorro.comer(); // Saída: Estou comendo
meuCachorro.latir(); // Saída: Au au!
```

No próximo capítulo, vamos aprender a manipular elementos HTML com JavaScript.

CAPÍTULO 4: MANIPULAÇÃO DE ELEMENTOS HTML



SELEÇÃO DE ELEMENTOS

JavaScript pode interagir com elementos HTML usando métodos como getElementById, getElementsByClassName, querySelector e querySelectorAll. **Exemplo:**

```
let elementoPorId = document.getElementById('meuId');
let elementosPorClasse =
document.getElementsByClassName('minhaClasse');
let primeiroElemento = document.querySelector('seletorCSS');
let todosElementos = document.querySelectorAll('seletorCSS');
```

MODIFICAÇÃO DE CONTEÚDO

Você pode modificar o conteúdo de um elemento usando a propriedade innerHTML. **Exemplo:**

```
let elemento = document.getElementById('meuElemento');
elemento.innerHTML = 'Novo conteúdo';
```

ADIÇÃO E REMOÇÃO DE ELEMENTOS

Para adicionar elementos, você pode usar métodos como appendChild ou insertAdjacentHTML. **Exemplo:**

```
let novoElemento = document.createElement('div');
novoElemento.innerHTML = 'Novo elemento';

let container = document.getElementById('meuContainer');
container.appendChild(novoElemento);
```

Para remover elementos, você pode usar o método removeChild. **Exemplo:**

```
let elementoParaRemover = document.getElementById
('elementoParaRemover');
elementoParaRemover.parentNode.removeChild(elementoPar
aRemover);
```

MANIPULAÇÃO DE ESTILOS E CLASSES

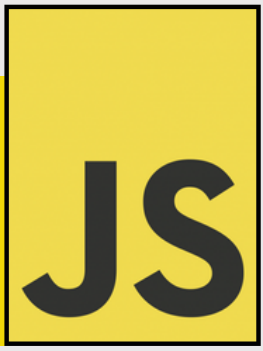
Você pode modificar estilos usando a propriedade style e adicionar ou remover classes usando classList.

Exemplo:

```
let elemento = document.getElementById('meuElemento');
elemento.style.color = 'blue';
elemento.classList.add('minhaClasse');
elemento.classList.remove('outraClasse');
```

No próximo capítulo, vamos explorar eventos e assincronia em JavaScript.

CAPÍTULO 5: EVENTOS E ASSINCRONIA



INTRODUÇÃO A EVENTOS

Eventos são ações que ocorrem no navegador, como cliques de mouse, pressionamento de teclas, entre outros. **Exemplo:**

```
let meuElemento = document.getElementById('meuElemento');

meuElemento.addEventListener('click', function() {
  console.log('O elemento foi clicado!');
});
```

ADICIONANDO LISTENERS

Você pode adicionar ouvintes de eventos para responder a ações do usuário. **Exemplo:**

```
let botao = document.getElementById('meuBotao');

function cliqueNoBotao() {
  console.log('O botão foi clicado!');
}

botao.addEventListener('click', cliqueNoBotao);
```

EVENTOS DE MOUSE, TECLADO, ETC.

Além de click, existem muitos outros eventos como mouseover, mouseout, keydown, keyup, etc. Você pode usar esses eventos para criar interações ricas.

Exemplo:

```
let elemento = document.getElementById('meuElemento');

elemento.addEventListener('mouseover', function() {
 console.log('O mouse está sobre o elemento!');
});
```

AJAX E REQUISIÇÕES ASSÍNCRONAS

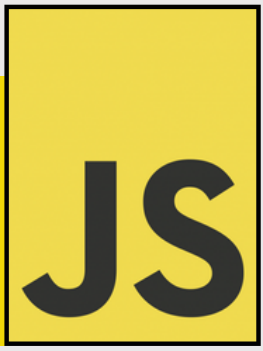
Você pode fazer requisições assíncronas para carregar dados de um servidor sem recarregar a página. **Exemplo:**

```
let xhr = new XMLHttpRequest();

xhr.open('GET', 'https://api.exemplo.com/dados', true);

xhr.onload = function() {
 if (xhr.status >= 200 && xhr.status < 400) {
  let resposta = JSON.parse(xhr.responseText);
  console.log(resposta);
 } else {
  console.error('Erro ao carregar dados');
 }
};
```

```
xhr.onerror = function() {
        console.error('Erro na requisição');
        };

xhr.send();
```

No próximo capítulo, vamos explorar como trabalhar com APIs em JavaScript.

CAPÍTULO 6: TRABALHANDO COM APIS



O QUE SÃO APIS

Uma API (Interface de Programação de Aplicativos) permite que diferentes sistemas se comuniquem entre si. No contexto web, geralmente se refere a um conjunto de endpoints que fornecem acesso a dados ou funcionalidades de um servidor.

CONSUMINDO APIS RESTFUL

Para consumir uma API RESTful, você pode usar a função fetch. **Exemplo:**

```
fetch('https://api.exemplo.com/dados')
 .then(response => response.json())
 .then(data => console.log(data))
 .catch(error => console.error('Erro:', error));
```

TRATAMENTO DE RESPOSTAS

Após obter uma resposta da API, é importante verificar o status para garantir que a requisição foi bem-sucedida.

TRATAMENTO DE RESPOSTAS

Exemplo:

```
fetch('https://api.exemplo.com/dados')
 .then(response => {
  if (!response.ok) {
   throw new Error('Erro na requisição');
  }
  return response.json();
 })
 .then(data => console.log(data))
 .catch(error => console.error('Erro:', error));
```

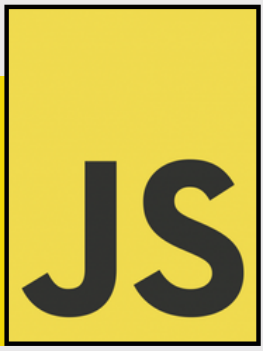
EXEMPLOS PRÁTICOS

Vamos ver um exemplo prático de como listar os usuários de uma API fictícia:

```
fetch('https://jsonplaceholder.typicode.com/users')
 .then(response => response.json())
 .then(users => {
  users.forEach(user => {
   console.log(`Nome: ${user.name}, Email: ${user.email}`);
  });
 })
 .catch(error => console.error('Erro:', error));
```

No próximo capítulo, vamos explorar exemplos práticos de como aplicar os conhecimentos adquiridos.

CAPÍTULO 7: EXEMPLOS PRÁTICOS



VALIDAÇÃO DE FORMULÁRIOS

Vamos criar uma função que valida um formulário simples:

```
function validarFormulario() {
 let nome = document.getElementById('nome').value;
 let email = document.getElementById('email').value;

 if (nome === '' || email === '') {
  alert('Por favor, preencha todos os campos');
  return false;
 }

 return true;
}
```

E no HTML:

```
<form onsubmit="return validarFormulario()">
 <label for="nome">Nome:</label>
 <input type="text" id="nome"><br>

 <label for="email">Email:</label>
 <input type="email" id="email"><br>
```

```
<input type="submit" value="Enviar">
</form>
```

MANIPULAÇÃO DE LISTAS

Vamos criar uma função que adiciona itens a uma lista:

```
function adicionarItem() {
 let item = document.getElementById('novoItem').value;
 let lista = document.getElementById('minhaLista');

 let novoItem = document.createElement('li');
 novoItem.appendChild(document.createTextNode(item));
 lista.appendChild(novoItem);

 document.getElementById('novoItem').value = '';
}
```

E no HTML:

```
<input type="text" id="novoItem">
<button onclick="adicionarItem()">Adicionar
Item</button>

<ul id="minhaLista">
 <li>Item 1</li>
 <li>Item 2</li>
 <li>Item 3</li>
</ul>
```

ANIMAÇÕES SIMPLES

Vamos criar uma animação simples de fade-in:

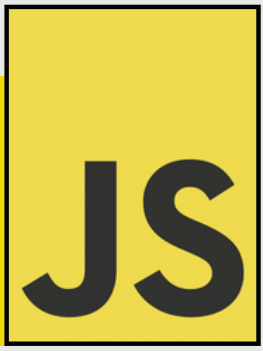
```
function fadeIn(elemento, duracao) {
 elemento.style.opacity = 0;
 let inicio = null;
 function animar(tempoAtual) {
  if (!inicio) inicio = tempoAtual;
  let progresso = tempoAtual - inicio;
  elemento.style.opacity = progresso / duracao;
  if (progresso < duracao) {
   requestAnimationFrame(animar);
  } }
 requestAnimationFrame(animar);
}
let meuElemento =
document.getElementById('elementoParaAnima
r');
fadeIn(meuElemento, 1000);
```

E no HTML:

```
<div id="elementoParaAnimar" style="width:
100px; height: 100px; background-color:
blue;"></div>
```

Este é um exemplo de como você pode aplicar os conceitos de JavaScript em situações práticas. Continue experimentando e criando projetos para aprimorar suas habilidades!

CONCLUSÃO

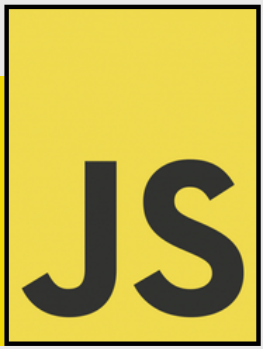


Parabéns! Agora você tem uma sólida compreensão dos conceitos fundamentais do JavaScript e como aplicá-los em situações práticas. Lembre-se de que a prática é essencial para se tornar proficiente nesta linguagem poderosa.

Continue explorando e construindo projetos para aprimorar suas habilidades. À medida que você avança, você pode se aprofundar em tópicos mais avançados, como programação assíncrona, manipulação de DOM mais complexa, frameworks e bibliotecas populares, entre outros.

Lembre-se sempre de consultar documentações e recursos online, e não hesite em participar de comunidades de desenvolvedores para aprender com os outros e obter ajuda quando precisar.
Espero que este eBook tenha sido útil para você. Boa sorte em sua jornada no mundo do JavaScript!

APÊNDICES



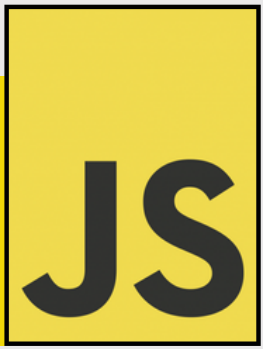
FERRAMENTAS E RECURSOS ÚTEIS

- Mozilla Developer Network (MDN): Excelente recurso de documentação para JavaScript e web em geral.
- W3Schools: Tutoriais e referências para várias tecnologias web.
- Codecademy: Oferece cursos interativos de JavaScript.
- Stack Overflow: Ótima comunidade para fazer e responder perguntas sobre programação.

GLOSSÁRIO DE TERMOS

- Variáveis: Espaços de memória onde você pode armazenar valores.
- Operadores: Símbolos que realizam operações em variáveis e valores.
- Funções: Blocos de código reutilizáveis que executam uma tarefa específica.

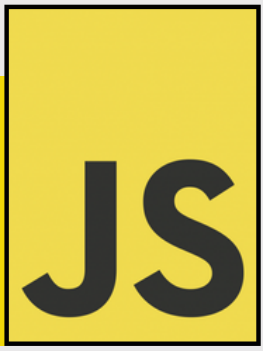
APÊNDICES



GLOSSÁRIO DE TERMOS

- Objetos: Coleções de pares chave-valor que representam dados ou funcionalidades.
- Classes: Modelos para criar objetos com propriedades e métodos.
- APIs: Conjuntos de endpoints que fornecem acesso a dados ou funcionalidades de um servidor.
- Eventos: Ações que ocorrem no navegador, como cliques de mouse ou pressionamentos de teclas.
- Requisições Assíncronas: Realização de operações sem bloquear a execução do código.
- DOM (Document Object Model): Representação estruturada de um documento HTML que JavaScript pode manipular.

SOBRE O AUTOR



Atualmente, estamos focados em explorar o emocionante campo da tecnologia, com ênfase em práticas DevOps.

Ao longo dos nossos estudos, pudemos mergulhar em diversas áreas, desde programação até gerenciamento de projetos, sempre buscando uma compreensão aprofundada do ecossistema tecnológico em constante evolução.

Este eBook sobre "Conceito e Preparação de Ambiente para Ensino DevOps" é o resultado de um intenso trabalho de pesquisa. Durante esse projeto, exploramos o mundo do DevOps, compreendendo suas bases, princípios e a importância de uma cultura colaborativa no desenvolvimento e operações de software.

Estou empolgado em compartilhar esse conhecimento com você e espero que este eBook seja um recurso valioso na sua jornada de aprendizado sobre DevOps.

OBRIGADO!

“Gostaria de expressar minha sincera gratidão a todos os leitores que dedicaram seu tempo a explorar este eBook. Espero que a jornada pelo mundo do JavaScript tenha sido tão gratificante para você quanto foi para mim ao criar este material.”