

MR. ROBOT

Código Mental Hacker

<Decifrando a Lógica de Programação/>

ADRIANO CORRÊA



.js

Sumário

Introdução

Visão geral da lógica de programação
Importância de aprender JavaScript
Conceitos Básicos

Estrutura de um programa JavaScript
Variáveis e Tipos de Dados
Operadores
Controle de Fluxo

Condicionais
Estruturas de Repetição
Funções

Declaração de Funções
Escopo e Contexto
Funções de Callback e Assíncronas
Estruturas de Dados

Arrays
Objetos
Manipulação de Estruturas de Dados
Programação Orientada a Objetos

Classes e Objetos
Herança
Polimorfismo
Manipulação de DOM
Seleção de Elementos
Eventos
Manipulação de Estilos e Conteúdos
Trabalhando com APIs

Requisições HTTP
Manipulação de Dados JSON
Promises e Async/Await
Depuração e Testes

Ferramentas de Depuração
Testes Unitários
Boas Práticas

Padrões de Código
Comentários e Documentação
Otimização de Código

1. Introdução

A Mente de um Programador

Programar é mais do que escrever código. É sobre resolver problemas, pensar logicamente e entender a máquina. JavaScript não é apenas uma linguagem, é uma ferramenta que pode transformar ideias em realidade.

Por Que JavaScript?

JavaScript é onnipresente. É executado em navegadores, servidores e até em dispositivos

IoT. Sua flexibilidade e ubiquidade o tornam uma escolha ideal para aprender programação.

01

Conceitos básicos

2. Fundamentos da Programação

Estrutura de um Programa

Todo programa em JavaScript começa com declarações básicas que definem variáveis, funções e a lógica do programa.

```
// Declaração de variável
let mensagem = "Olá, mundo!";
```

```
// Função simples
function saudacao() {
  console.log(mensagem);
}
```

```
// Chamada de função
saudacao();
```

Variáveis e Tipos de Dados

Em JavaScript, podemos definir variáveis usando `let`, `const` ou `var` (embora `var` seja menos recomendado)

```
let nome = "Elliot"; // String
const idade = 28; // Número
let ativo = true; // Booleano
let habilidades = ["hacking", "programação"]; // Array
let usuario = { nome: "Elliot", idade: 28 }; // Objeto
```

Operadores

Operadores são usados para realizar operações em variáveis e valores.

```
let a = 10;
let b = 5;
```

```
// Operadores aritméticos
let soma = a + b;
let diferenca = a - b;
```

```
// Operadores de comparação
let igual = (a === b);
let maior = (a > b);
```

```
// Operadores lógicos
let e = (a > b && b > 0);
let ou = (a > b || b < 0);
```

02

Controle de fluxo

Estruturas Condicionais

As condicionais controlam o fluxo do código com base em condições específicas.

```
let idade = 28;

if (idade >= 18) {
  console.log("Adulto");
} else {
  console.log("Menor de idade");
}
```

Laços de Repetição

Repetem blocos de código enquanto uma condição for verdadeira.

```
for (let i = 0; i < 5; i++) {
  console.log(`Iteração ${i}`);
}
```

```
let count = 0;
while (count < 5) {
  console.log(`Contagem ${count}`);
  count++;
}
```

03

Funções

Definindo Funções

Funções encapsulam código para reutilização e organização.

```
function saudacao(nome) {  
    return `Olá, ${nome}!`;  
}  
  
console.log(saudacao("Elliot"));
```

Escopo e Contexto

O escopo determina a visibilidade das variáveis.

```
let global = "Global";  
  
function mostrarEscopo() {  
    let local = "Local";  
    console.log(global); // Acessa variável global  
    console.log(local); // Acessa variável local  
}  
  
mostrarEscopo();  
// console.log(local); // Erro: local não está definida fora da função
```

Funções de Callback e Assíncronas

JavaScript é assíncrono e orientado a eventos.

```
function processarDados(callback) {  
    setTimeout(() => {  
        let dados = "Dados processados";  
        callback(dados);  
    }, 1000);  
}  
  
function exibirDados(dados) {  
    console.log(dados);  
}  
  
processarDados(exibirDados);
```

04

Estrutura de dados

5. Estruturas de Dados

Arrays

Arrays são coleções de itens ordenados.

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

numeros.forEach(numero => {
  console.log(numero);
});
```

Objetos

Objetos são coleções de pares chave-valor.

```
let usuario = {
  nome: "Elliot",
  idade: 28,
  profissao: "Engenheiro de Cibersegurança"
};

console.log(usuario.nome);
console.log(usuario["idade"]);
```

Manipulação de Dados

Adicionar, remover e modificar elementos.

```
let habilidades = ["hacking", "programação"];
habilidades.push("engenharia social"); // Adiciona ao final
habilidades.splice(1, 1); // Remove 1 elemento a partir do índice 1
```

```
let usuario = {
  nome: "Elliot",
  idade: 28
};
```

```
usuario.profissao = "Engenheiro de Cibersegurança"; // Adiciona uma nova propriedade
delete usuario.idade; // Remove a propriedade idade
```

05

Programação orientada a objetos

Conceitos Básicos

Classes são modelos para criar objetos.

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

  apresentar() {
    return `Olá, meu nome é ${this.nome} e tenho ${this.idade} anos.`;
  }
}

let elliot = new Pessoa("Elliot", 28);
console.log(elliot.apresentar());
```

Herança

Herança permite criar novas classes baseadas em classes existentes.

```
class Hacker extends Pessoa {
  constructor(nome, idade, alias) {
    super(nome, idade);
    this.alias = alias;
  }

  hackear() {
    return `${this.alias} está hackeando...`;
  }
}

let mrRobot = new Hacker("Elliot", 28, "Mr. Robot");
console.log(mrRobot.apresentar());
console.log(mrRobot.hackear());
```

Polimorfismo

Métodos em classes derivadas podem ter implementações diferentes.

```
class Animal {
  fazerSom() {
    console.log("O animal faz um som");
  }
}

class Cachorro extends Animal {
  fazerSom() {
    console.log("O cachorro late");
  }
}

class Gato extends Animal {
  fazerSom() {
    console.log("O gato mia");
  }
}
```

06

Manipulando o DOM

Selecionando Elementos

Selecionar elementos no DOM é essencial para interações dinâmicas.

```
let titulo = document.querySelector("h1");
console.log(titulo.textContent);
```

Eventos

Eventos são ações que ocorrem na página.

```
let botao = document.querySelector("button");

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

Alterando Estilos e Conteúdos

Podemos alterar estilos e conteúdos de elementos.

```
let titulo = document.querySelector("h1");

titulo.textContent = "Novo Título";
titulo.style.color = "red";
```

07

Trabalhando com APIs

Requisições HTTP

Interagir com servidores via HTTP.

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

Manipulação de JSON

JSON é um formato comum para transferência de dados.

```
let jsonData = '{"nome": "Elliot", "idade": 28}';
let usuario = JSON.parse(jsonData);

console.log(usuario.nome);
```

Promises e Async/Await

Promises ajudam a lidar com operações assíncronas.

```
function obterDados() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve("Dados obtidos");
    }, 1000);
  });
}
```

```
obterDados()
  .then(dados => {
    console.log(dados);
  })
  .catch(error => {
    console.error("Erro:", error);
  });
```

```
// Usando async/await
async function exibirDados() {
  try {
    let dados = await obterDados();
    console.log(dados);
  } catch (error) {
    console.error("Erro:", error);
  }
}

exibirDados();
```

08

Depuração e testes

Ferramentas de Depuração

Use o console do navegador e ferramentas como o debugger.

```
    let valor = 10;
    console.log(valor);
    debugger; // Pausa a execução para inspeção
    valor += 5;
    console.log(valor);
```

Testes Unitários

Testes garantem que o código funcione conforme esperado.

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

console.assert(soma(2, 3) === 5, "Teste falhou");
console.assert(soma(-1, 1) === 0, "Teste falhou");
```

09

Boas práticas

Padrões de Código

Consistência no código melhora a legibilidade e manutenção.

```
// Evite  
let a=10;
```

```
// Melhor  
let a = 10;
```

Comentários e Documentação

Comentários claros ajudam na compreensão e manutenção do código.

```
// Calcula a soma de dois números  
function soma(a, b) {  
    return a + b;  
}
```

Otimização

Escrever código eficiente melhora o desempenho da aplicação.

```
// Evite loops desnecessários  
for (let i = 0; i < 100; i++) {  
    console.log(i);  
}
```

```
// Melhor usar métodos de array quando possível  
[...Array(100).keys()].forEach(i => console.log(i));
```

Este livro é seu guia para entender a lógica de programação com JavaScript, explorando conceitos fundamentais até tópicos avançados. Cada capítulo é projetado para aprofundar seu conhecimento e aprimorar suas habilidades. Boa leitura e boa programação.