JAVASCRIPT

Seminário 01: Linguagem Origem

Grupo:

Erica Kathlen, Igor Dias, João Marcelo,
João Guilherme, Rafael Alexander.

TÓPICOS

SUMÁRIO

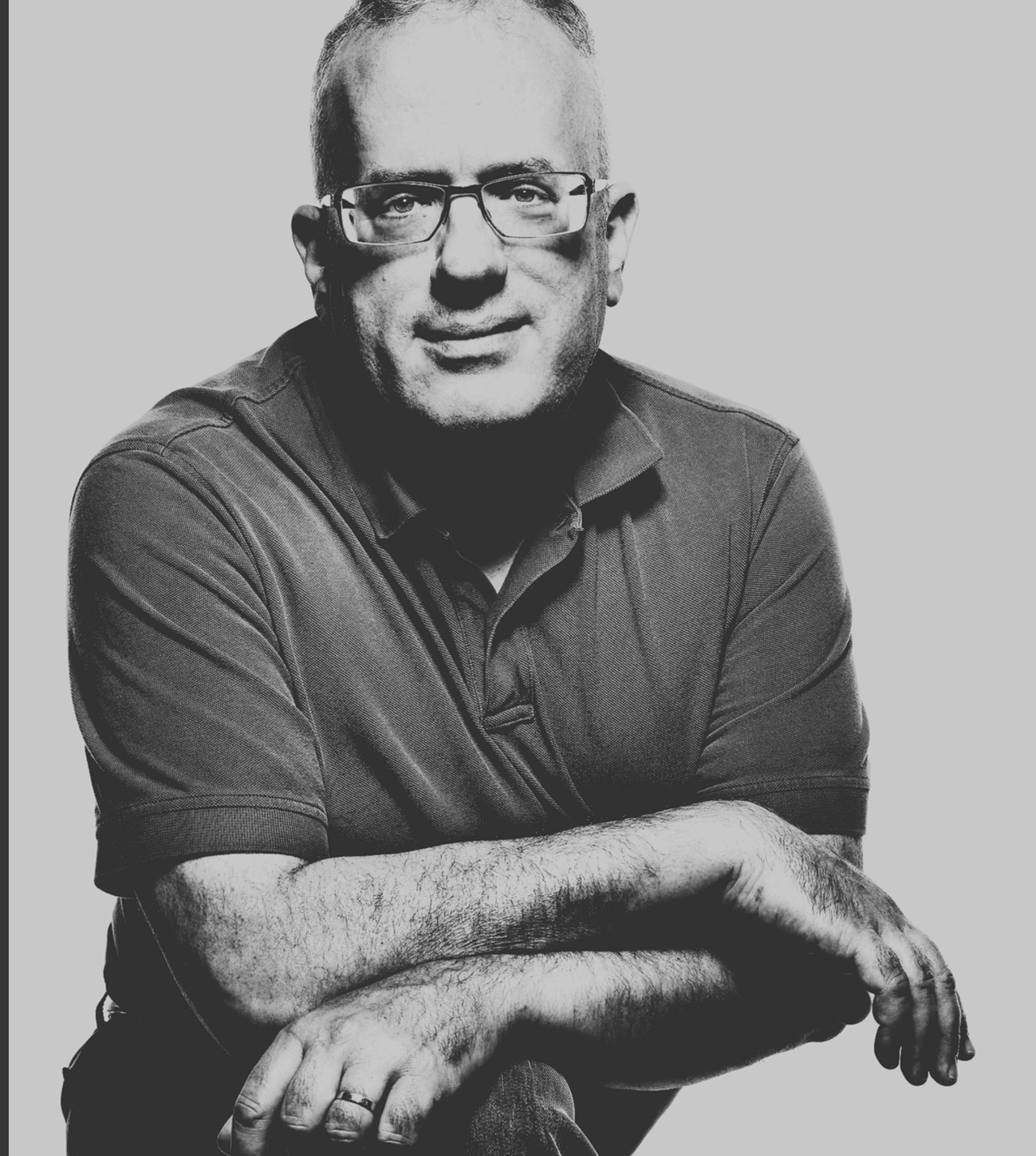
- História;
- Paradigmas de programação suportados;
- Tipos de aplicações; Frameworks e Bibliotecas;
- IDEs;
- Como fazer um Hello World;

HISTÓRIA

JavaScript foi criado em 1995 por Brendan Eich, da Netscape, inicialmente em apenas 10 dias. Seu primeiro nome foi Mocha, depois LiveScript, e finalmente JavaScript.

Apesar do nome, não tem ligação direta com Java — foi uma jogada de marketing da época.

Com o tempo, se tornou a linguagem padrão da web, principalmente com a padronização pelo ECMAScript (ECMA-262) em 1997. Hoje, está presente em frontend, backend (Node.js), mobile e até IoT



PARADIGMAS DE PROGRAMAÇÃO

APLICAÇÕES EM DESTAQUES

1. Aplicações Web Frontend
2. Backend
3. Mobile
4. Desktop
5. IoT

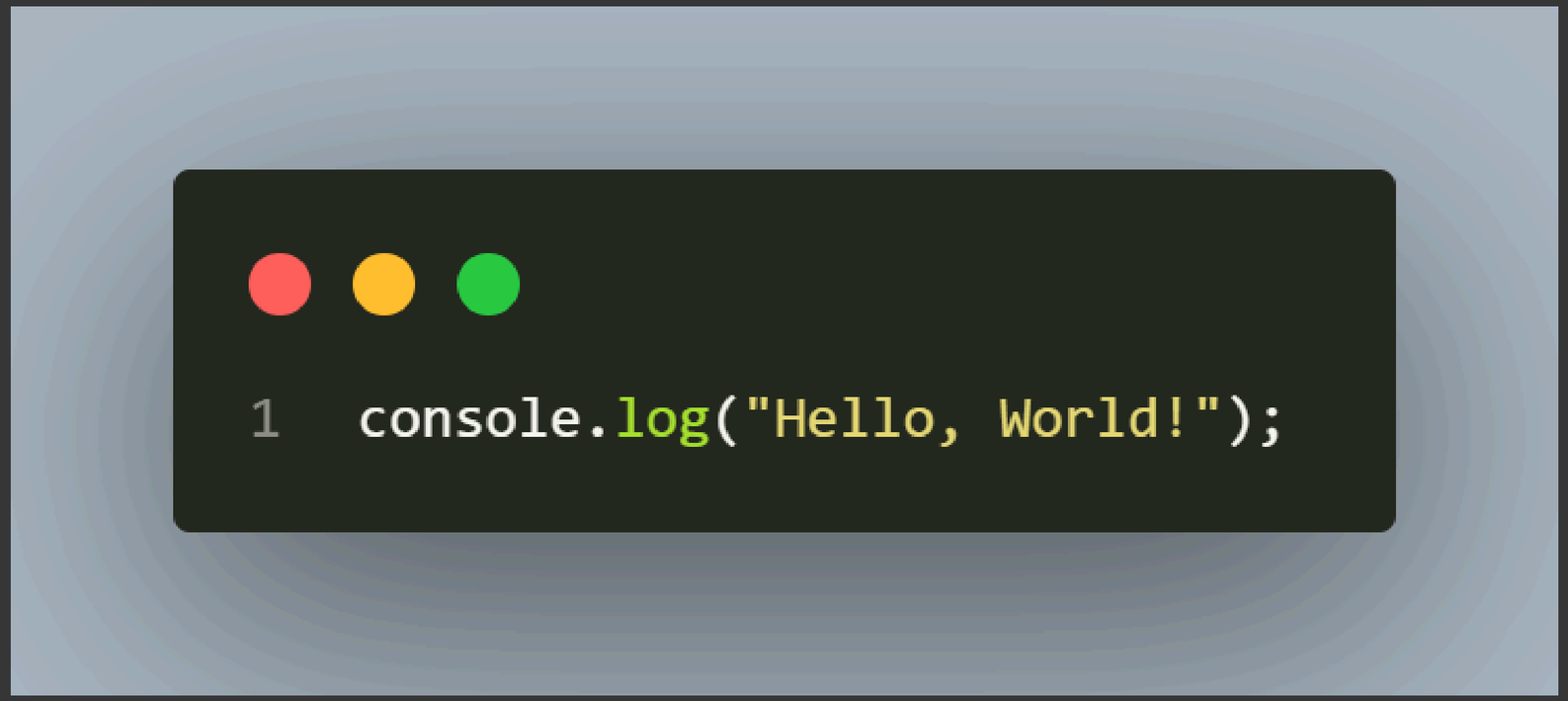
FRAMEWORKS E BIBLIOTECAS

1. Frontend: React, Angular, Vue.js, Svelte.
2. Backend: Express.js, NestJS, Koa.
3. Fullstack: Next.js, Nuxt.js, Meteor.
4. Mobile: React Native, Ionic, NativeScript.
5. Testes: Jest, Mocha, Jasmine.

IDES RECOMENDADAS

1. Visual Studio Code (VSCode)
2. WebStorm (JetBrains)
3. Atom
4. Sublime Text
5. Replit / Codesandbox

HELLO, WORLD



Seminário 02 - Linguagem origem

JAVASCRIPT

Grupo:

Erica Kathlen, Igor Dias, João Marcelo,
João Guilherme, Rafael Alexander.

TIPOS DE DADOS

- Primitivos;
- Compostos.

```javascript
// --- TIPOS DE DADOS ---

// Tipos Primitivos
let idade = 25; // Number (IEEE 754)
let nome = "Maria"; // String (UTF-16)
let ativo = true; // Boolean
let projeto = null; // Null (ausência de objeto)
let tarefa; // Undefined (valor não atribuído)

// Tipo Composto (Objeto)
let pessoa = {
    nome: "Carlos",
    idade: 30
};

// Array (um tipo de objeto) - dinâmico
let tecnologias = ["JavaScript", "Web Workers"];
tecnologias.push("SharedArrayBuffer"); // Vetor é dinâmico

console.log(`A tecnologia principal é ${tecnologias[0]}`); // Acesso por índice numérico
```

USO DE PONTEIROS E REFERÊNCIAS

- Passagem por Valor;
- Passagem por Referência;

```
// --- PONTEIRO E REFERENCIAS ---

// Passagem por VALOR (primitivos)
let a = 10;
let b = a; // 'b' é uma cópia de 'a'
b = 20;
console.log(`a = ${a}, b = ${b}`); // Saída: a = 10, b = 20

// Passagem por REFERÊNCIA (objetos)
let objA = { valor: 10 };
let objB = objA; // 'objB' aponta para o MESMO objeto que 'objA'
objB.valor = 20;
console.log(`objA.valor = ${objA.valor}, objB.valor = ${objB.valor}`); // Saída: objA.valor = 20, objB.valor = 20
```

PALAVRAS-CHAVE E RESERVADAS

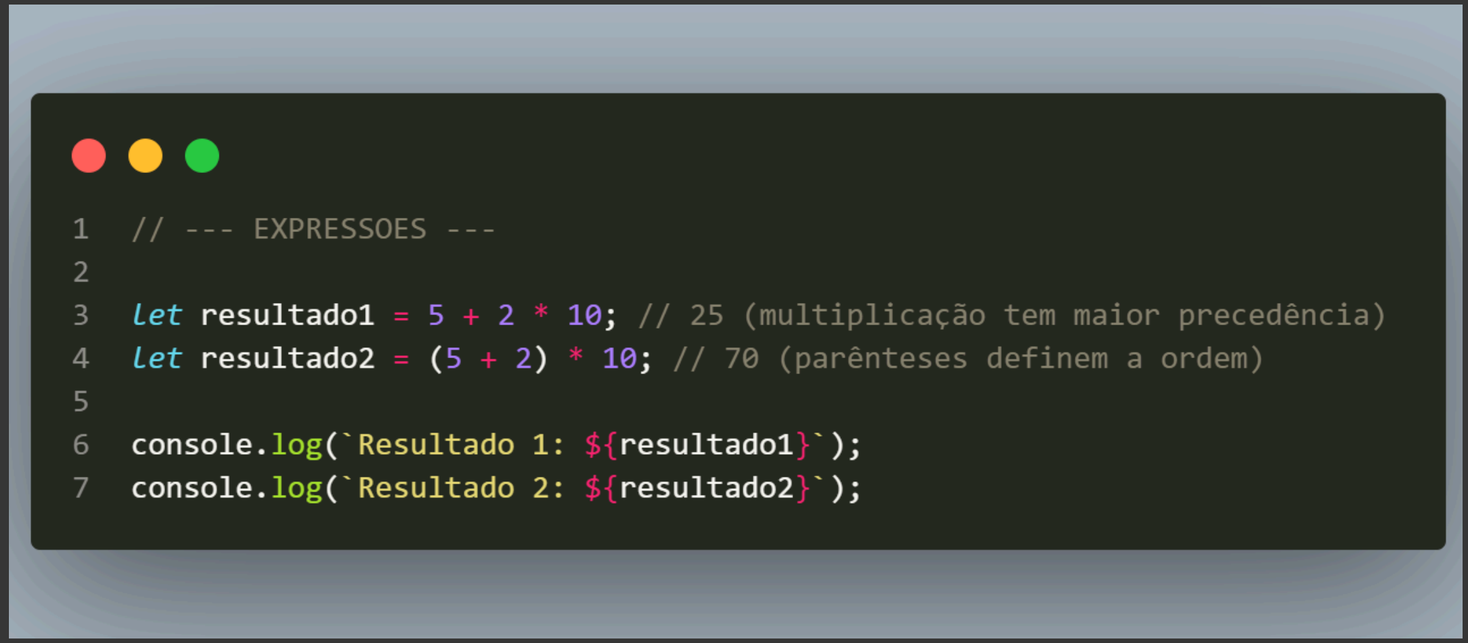
Palavras-chave como ***let, const, if, for*** e ***function*** são reservadas pela linguagem e não podem ser usadas como nomes de variáveis.

```
// --- PALAVRAS-CHAVES E VARIAVEIS ---

function escopoExemplo() {
  if (true) {
    var nomeVar = "Sou visível em toda a função"; // Escopo de função
    let nomeLet = "Sou visível apenas neste bloco"; // Escopo de bloco
    const PI = 3.14; // Escopo de bloco, não pode ser reatribuída
  }

  console.log(nomeVar); // Funciona
  // console.log(nomeLet); // Erro! nomeLet não está definido aqui.
}

escopoExemplo();

// let for = "erro"; // Erro de sintaxe, 'for' é uma palavra reservada.
```

EXPRESSÕES

Expressões são avaliadas para produzir um valor. A ordem de avaliação é definida pela precedência e associatividade.

A sobrecarga de operadores não é suportada nativamente em JS.



```
// --- EXPRESSOES ---

let resultado1 = 5 + 2 * 10; // 25 (multiplicação tem maior precedência)
let resultado2 = (5 + 2) * 10; // 70 (parênteses definem a ordem)

console.log(`Resultado 1: ${resultado1}`);
console.log(`Resultado 2: ${resultado2}`);
```

ESTRUTURA DE CONTROLE

- Condicional:
  - ***if...else;***
  - ***switch;***

- Repetição:
  - ***for;***
  - ***while;***
  - ***for...of;***

```
// --- ESTRUTURA DE CONTROLE ---

// Condicional (if/else)
let temperatura = 25;
if (temperatura > 30) {
  console.log("Está calor!");
} else {
  console.log("Temperatura agradável.");
}

// Repetição (for...of)
let numeros = [10, 20, 30];
console.log("Iterando sobre o array:");

for (const numero of numeros) {
  console.log(numero);
}
```

JAVASCRIPT

Seminário 03 - Linguagem origem

TÓPICOS

SUMÁRIO

1. Subprogramas;
2. Concorrência;
3. Tratamento de Exceções;
4. Tratamento de Eventos;
5. Paradigmas;

SUBPROGRAMAS

- **Declaração e Chamada:** palavra-chave = *function*
- **Passagem de Parâmetros:** '*passagem por valor*' e '*passagem por referência*'.

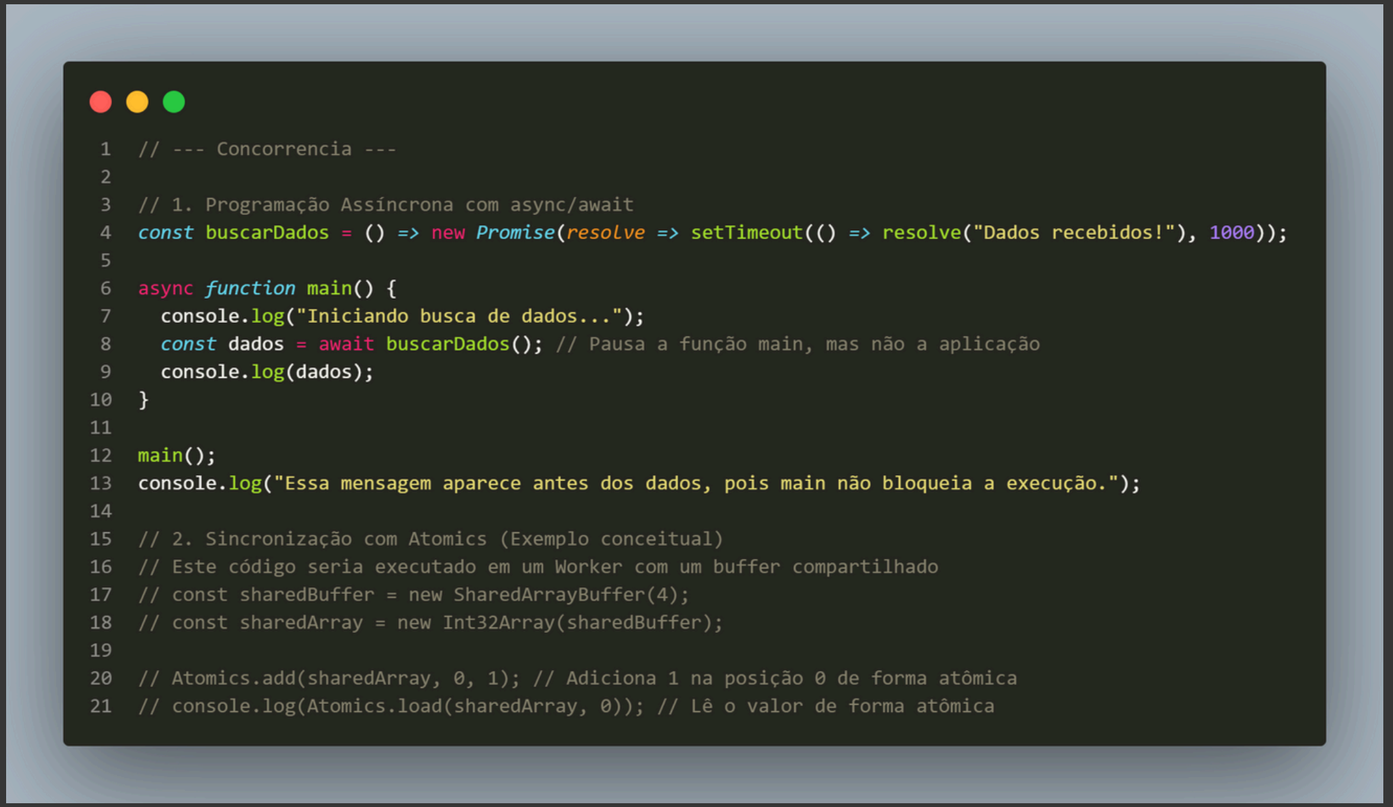
- **Funções Lambda:** *Arrow Functions* (=>).
- **Subprogramas Delegados:** *callbacks.*

- **Co-rotinas:** Generator Functions (*function**) e o comando *yield.*
- **Programação Genérica:** JS é uma linguagem de tipagem dinâmica.

```
// --- Subprogramas ---

// Declaração Padrão
function somar(a, b) {
  return a + b;
}

// Expressão de Função (Lambda / Arrow Function)
const multiplicar = (a, b) => a * b;

// Delegate (usando uma função como callback)
function calcular(a, b, operacao) {
  return operacao(a, b);
}
console.log(`Resultado do delegate: ${calcular(10, 5, multiplicar)}`);

// Co-rotina (Generator Function)
function* contador() {
  yield 1; // Pausa a execução e retorna 1
  yield 2; // Pausa a execução e retorna 2
  return 3;
}

const gen = contador();
console.log(`Co-rotina (yield 1): ${gen.next().value}`);
console.log(`Co-rotina (yield 2): ${gen.next().value}`);
```

CONCORRÊNCIA E SINCRONIZAÇÃO

- **Programação Assíncrona;**
  - *async/await;*
- **Mecanismos de Sincronização;**
  - *SharedArrayBuffer;*
  - *Atomics;*



```
// --- Concorrencia ---

// 1. Programação Assíncrona com async/await
const buscarDados = () => new Promise(resolve => setTimeout(() => resolve("Dados recebidos!"), 1000));

async function main() {
  console.log("Iniciando busca de dados...");
  const dados = await buscarDados(); // Pausa a função main, mas não a aplicação
  console.log(dados);
}

main();
console.log("Essa mensagem aparece antes dos dados, pois main não bloqueia a execução.");

// 2. Sincronização com Atomics (Exemplo conceitual)
// Este código seria executado em um Worker com um buffer compartilhado
// const sharedBuffer = new SharedArrayBuffer(4);
// const sharedArray = new Int32Array(sharedBuffer);

// Atomics.add(sharedArray, 0, 1); // Adiciona 1 na posição 0 de forma atômica
// console.log(Atomics.load(sharedArray, 0)); // Lê o valor de forma atômica
```

TRATAMENTO DE EXCEÇÕES

- **Tipos de Erros:**
  - *ReferenceError;*
  - *TypeError;*

- **Tratamento:**
  - ***try:*** Contém o código que pode gerar uma exceção;
  - ***catch:*** Captura a exceção, se ocorrer, permitindo um tratamento adequado;
  - ***finally;***

```
1  // --- Excecoes ---
2
3  function dividir(a, b) {
4    if (b === 0) {
5      // Lançando uma nova exceção
6      throw new Error("Divisão por zero não é permitida!");
7    }
8    return a / b;
9  }
10
11 try {
12   console.log("Tentando dividir 10 por 2...");
13   const resultado = dividir(10, 2);
14   console.log(`Resultado: ${resultado}`);
15
16   console.log("\nTentando dividir 10 por 0...");
17   dividir(10, 0); // Esta linha vai gerar uma exceção
18
19 } catch (erro) {
20   // Capturando e tratando a exceção
21   console.error(`Erro capturado: ${erro.message}`);
22 } finally {
23   // Este bloco sempre executa
24   console.log("\nBloco de tratamento finalizado.");
25 }
```

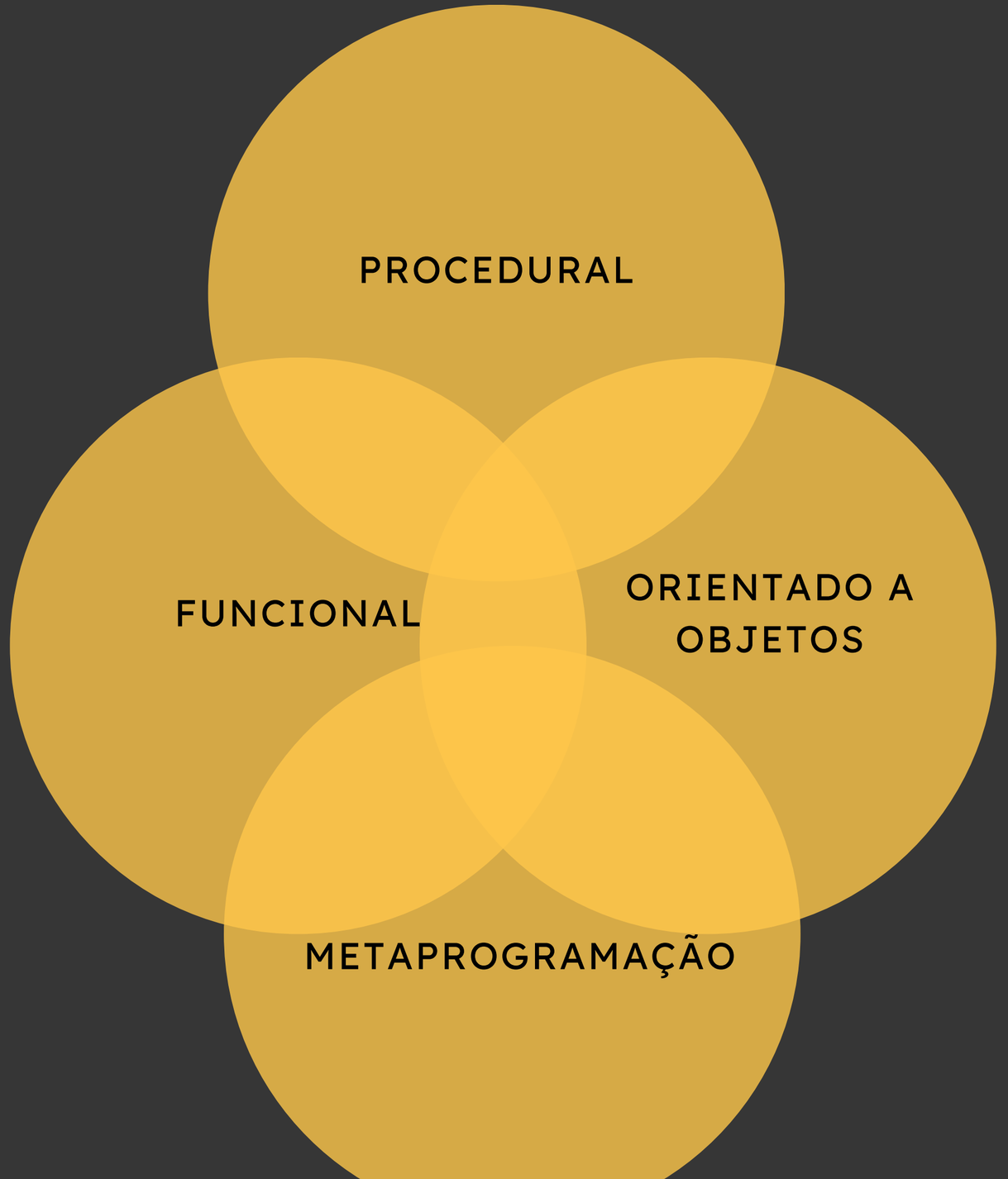
TRATAMENTO DE EVENTOS

- **Tipos de Eventos:**
  - *click ou keydown;*
  - *fetch;*
  - *setTimeout;*
  - *onmessage;*

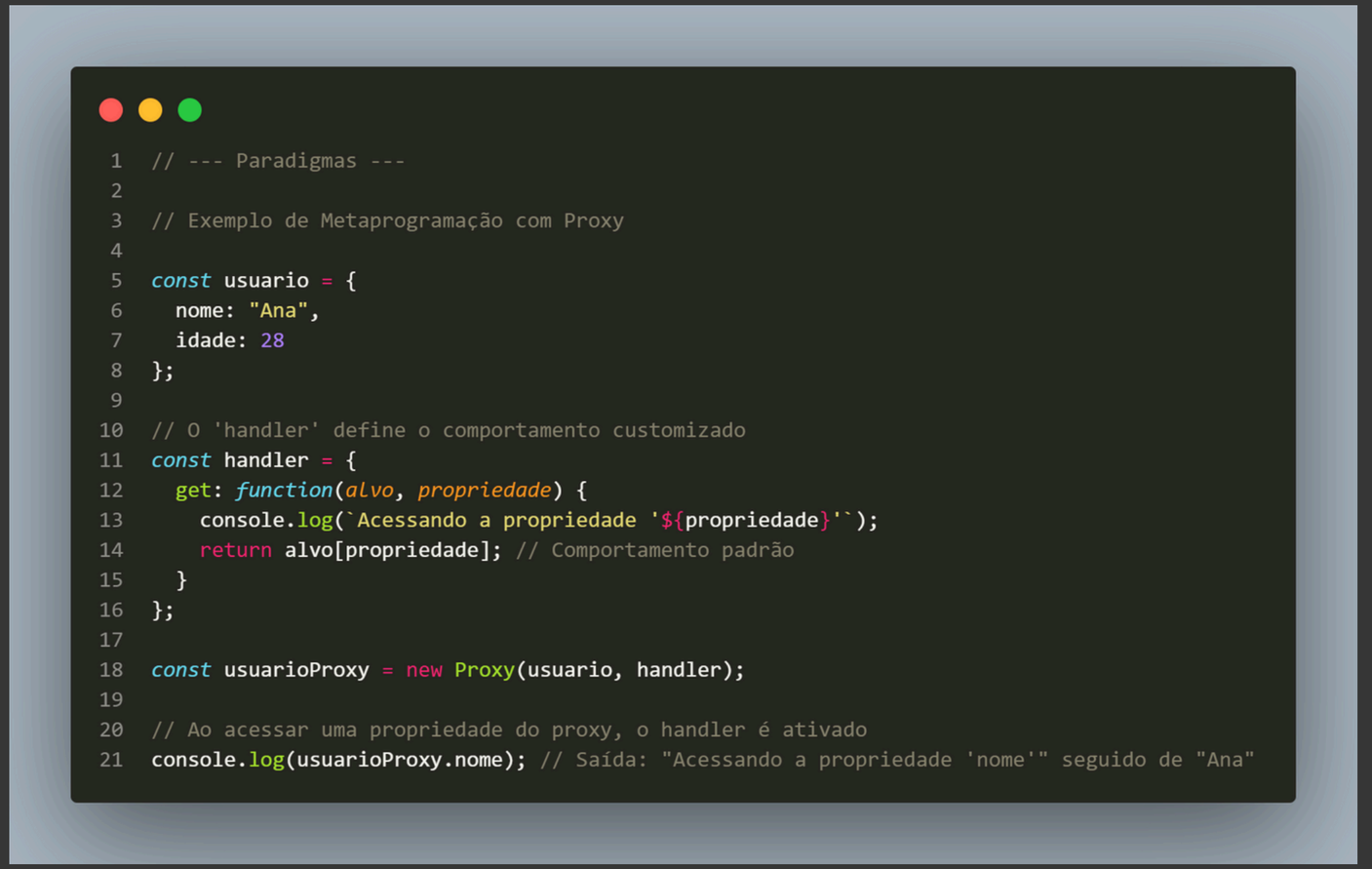
- **Implementação:**
  - *'event listeners'*

```
// --- Eventos ---

// Exemplo genérico com Timer
console.log("Um evento será disparado em 2 segundos...");

setTimeout(() => {
  // Esta função é o 'event handler' para o evento de 'timeout'
  console.log("Evento de timer disparado!");
}, 2000); // 2000 milissegundos
```

ASPECTOS E PARADIGMAS



EXEMPLO DE CÓDIGO



```
// --- Paradigmas ---

// Exemplo de Metaprogramação com Proxy

const usuario = {
  nome: "Ana",
  idade: 28
};

// O 'handler' define o comportamento customizado
const handler = {
  get: function(alvo, propriedade) {
    console.log(`Acessando a propriedade '${propriedade}'`);
    return alvo[propriedade]; // Comportamento padrão
  }
};

const usuarioProxy = new Proxy(usuario, handler);

// Ao acessar uma propriedade do proxy, o handler é ativado
console.log(usuarioProxy.nome); // Saída: "Acessando a propriedade 'nome'" seguido de "Ana"
```

OBRIGADO
PELA ATENÇÃO!