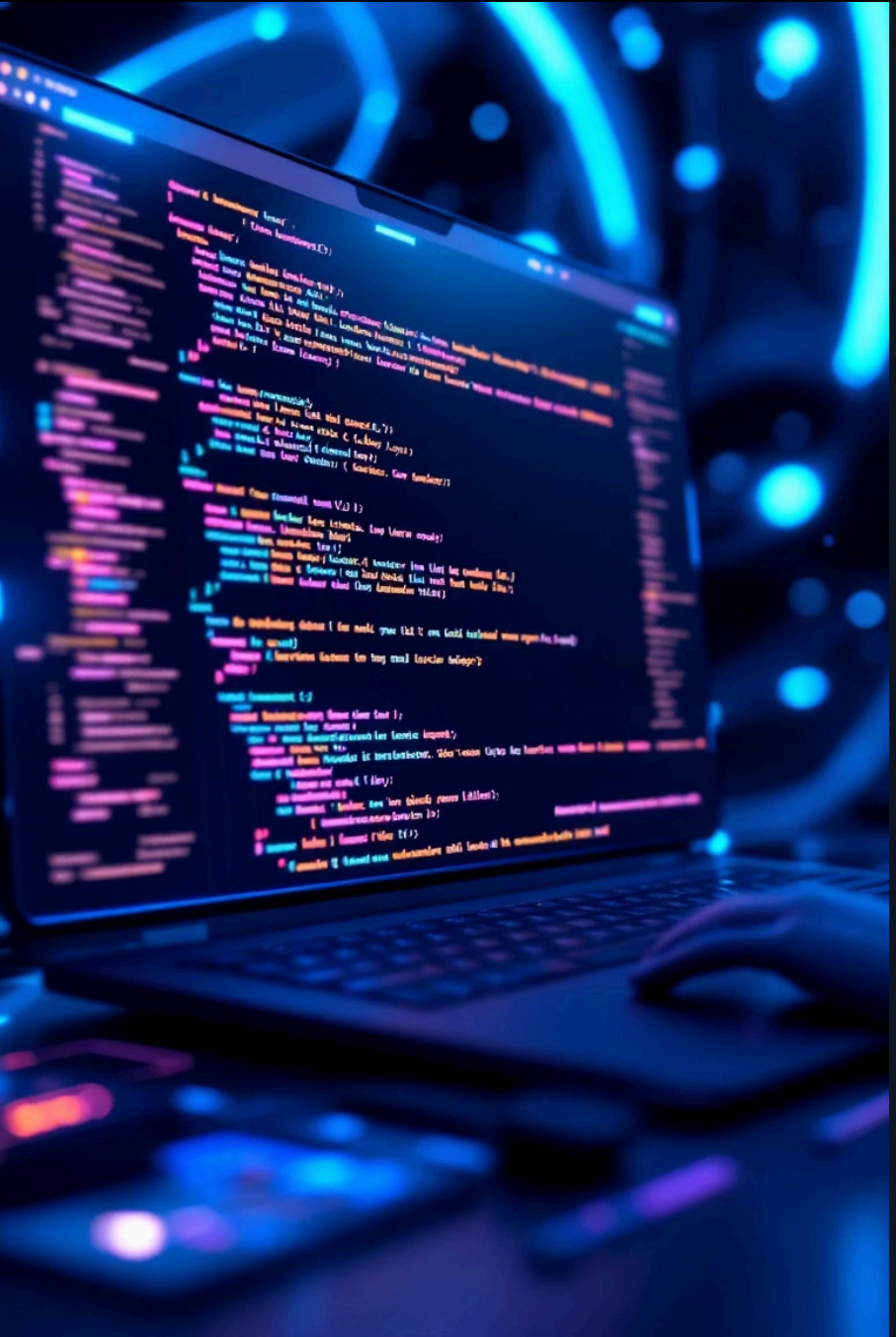


Linguagem de Programação JavaScript

03 - Laços de repetição, e funções

O Loop While: Funcionamento

O laço while executa um bloco de código enquanto uma condição específica permanece verdadeira. Compreender seu fluxo de execução é essencial para programar com eficiência.

01

Inicialização

Declare e inicialize as variáveis de controle antes do início do laço

02

Condição

A expressão entre parênteses é avaliada como verdadeira ou falsa

03

Execução

Se verdadeira, o bloco entre chaves é executado; se falsa, o laço termina

04

Alteração

Modifique a variável de controle para garantir que a condição se torne falsa eventualmente

05

Repetição

O processo retorna à verificação da condição, criando o ciclo de repetição

Exemplo Prático do While

Código

```
let contador = 0;

while (contador < 3) {
  console.log("Contador é: "
    + contador);
  contador++;
}

console.log("Loop finalizado.");
```

Resultado

O programa exibirá as seguintes mensagens:

- Contador é: 0
- Contador é: 1
- Contador é: 2
- Loop finalizado.

O laço termina quando o contador atinge o valor 3, tornando a condição falsa.

O Loop For: Estrutura e Componentes

O laço for oferece uma sintaxe mais compacta, agrupando inicialização, condição e atualização em uma única linha.

1

Expressão Inicial

Executada uma única vez antes do início do laço. Utilizada para declarar e inicializar a variável de controle.

Exemplo: let i = 0;

2

Condição

Avaliada antes de cada iteração. Se verdadeira, o código é executado; se falsa, o laço é encerrado.

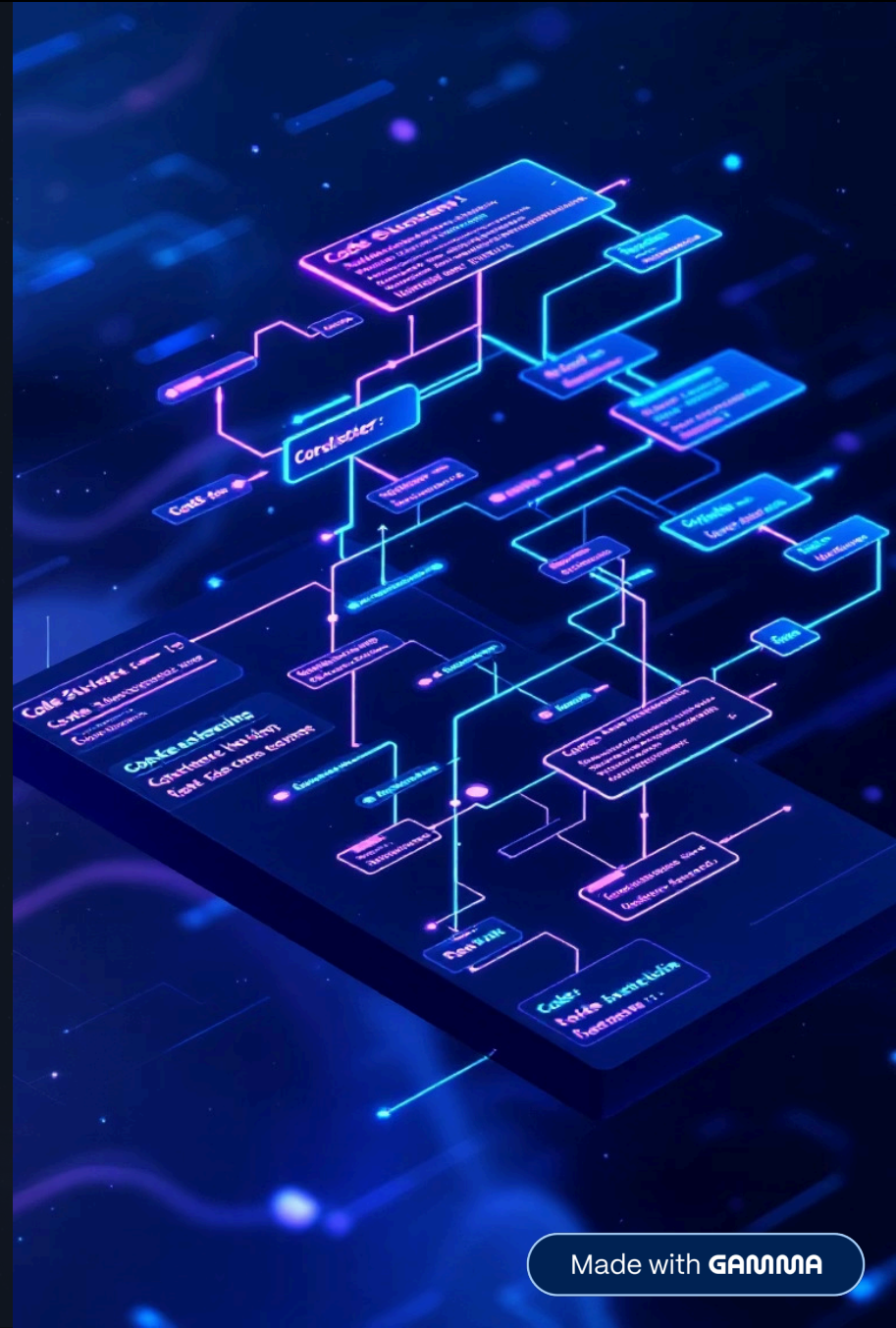
Exemplo: i < 10;

3

Expressão Final

Executada ao final de cada iteração. Normalmente utilizada para incrementar ou decrementar a variável de controle.

Exemplo: i++;



Sintaxe Completa do For

Estrutura Básica

```
for (expressaoInicial;
    condicao;
    expressaoFinal) {
  // Código executado
  // em cada iteração
}
```

Exemplo de Aplicação

Para exibir números de 0 a 4:

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

Este laço imprime os valores 0, 1, 2, 3 e 4 no console, demonstrando o controle preciso sobre as iterações.

O Laço Do...While

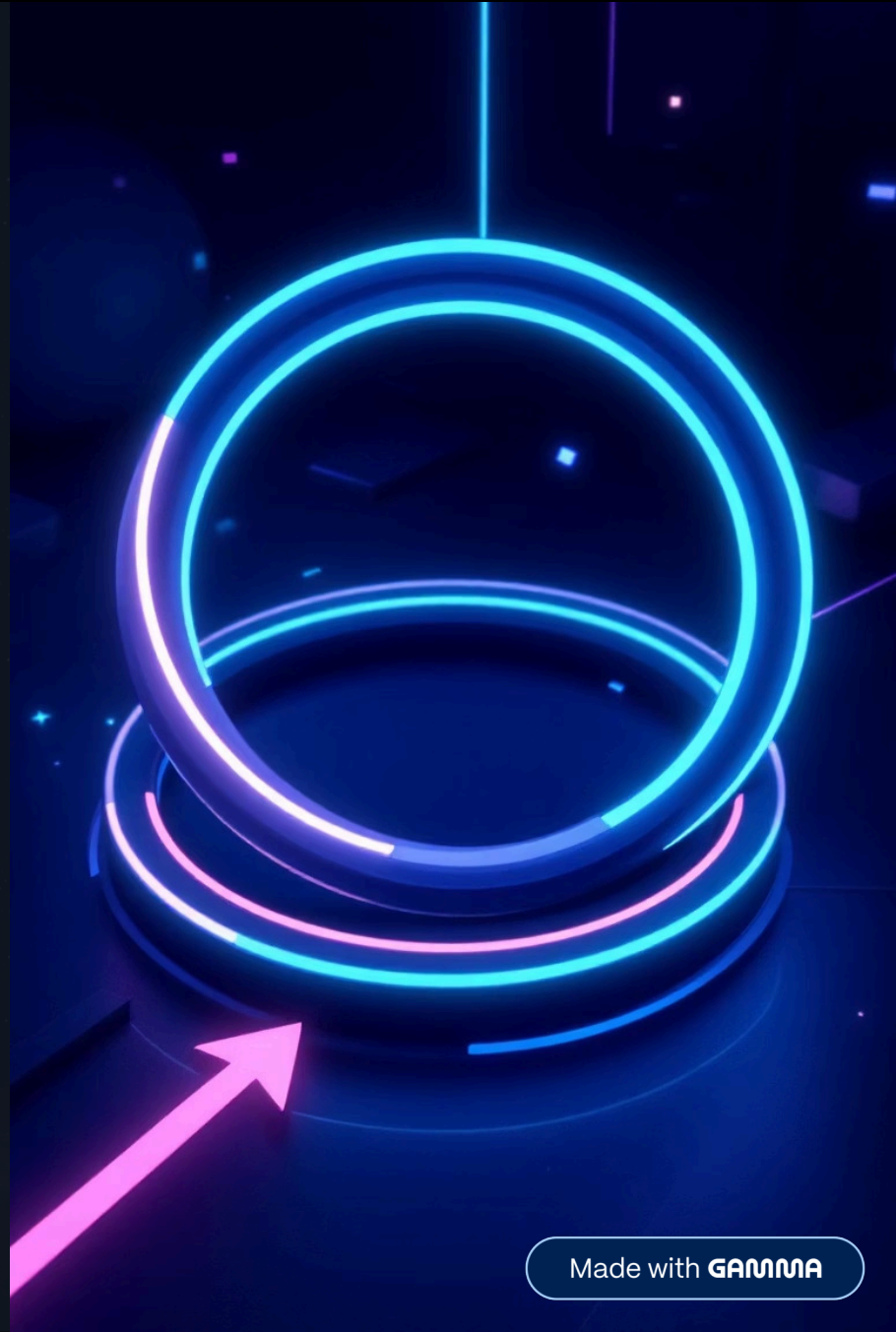
A estrutura do...while garante que o bloco de código seja executado ao menos uma vez, pois a condição é verificada após a execução.

```
let quantidade_participantes = 0;

do {
  console.log("Até o momento a quantidade de participantes é "
    + quantidade_participantes);
  quantidade_participantes += 1;
} while(quantidade_participantes < 20);

console.log("O limite de participantes foi atingido");
```

Esta estrutura é especialmente útil quando precisamos executar uma operação antes de verificar se ela deve continuar, como em validações ou processamentos iniciais obrigatórios.



Funções em JavaScript

Funções são blocos de código reutilizáveis que executam tarefas específicas. Elas organizam o código e permitem a reutilização de lógica.

Declaração de Função

Utilizam a palavra-chave function e podem ser chamadas antes de sua definição no código.

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

Expressões de Função

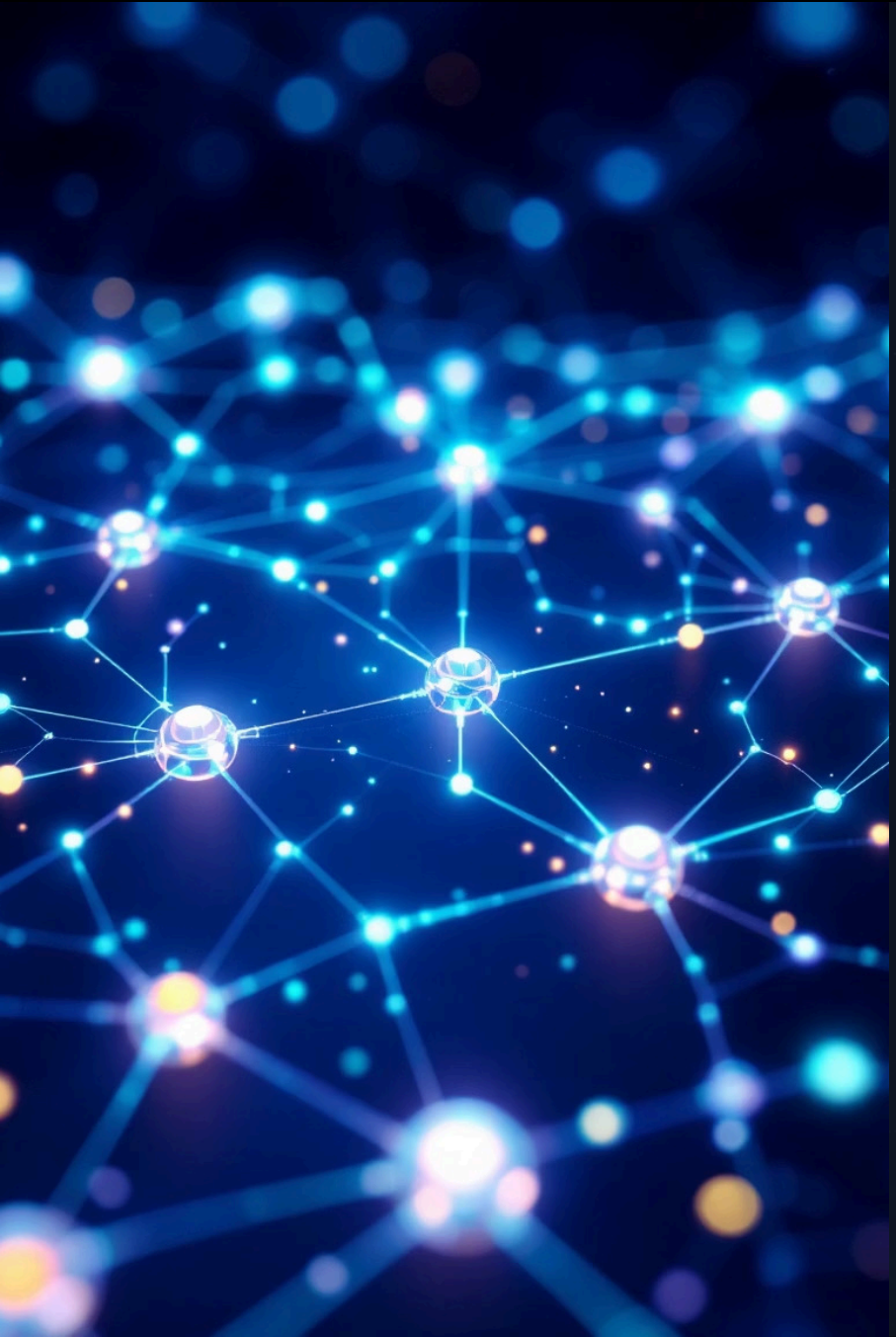
Incluem funções de seta (=>) e são atribuídas a variáveis.

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

Funções com Parâmetros

Recebem argumentos para processar valores dinâmicos.

```
function saudacao(nome) {
 console.log("Olá, " + nome);
}
saudacao("Mundo");
```



Operações Assíncronas

JavaScript permite executar operações sem bloquear o fluxo principal do programa, possibilitando maior eficiência e responsividade.

Natureza Não Bloqueante

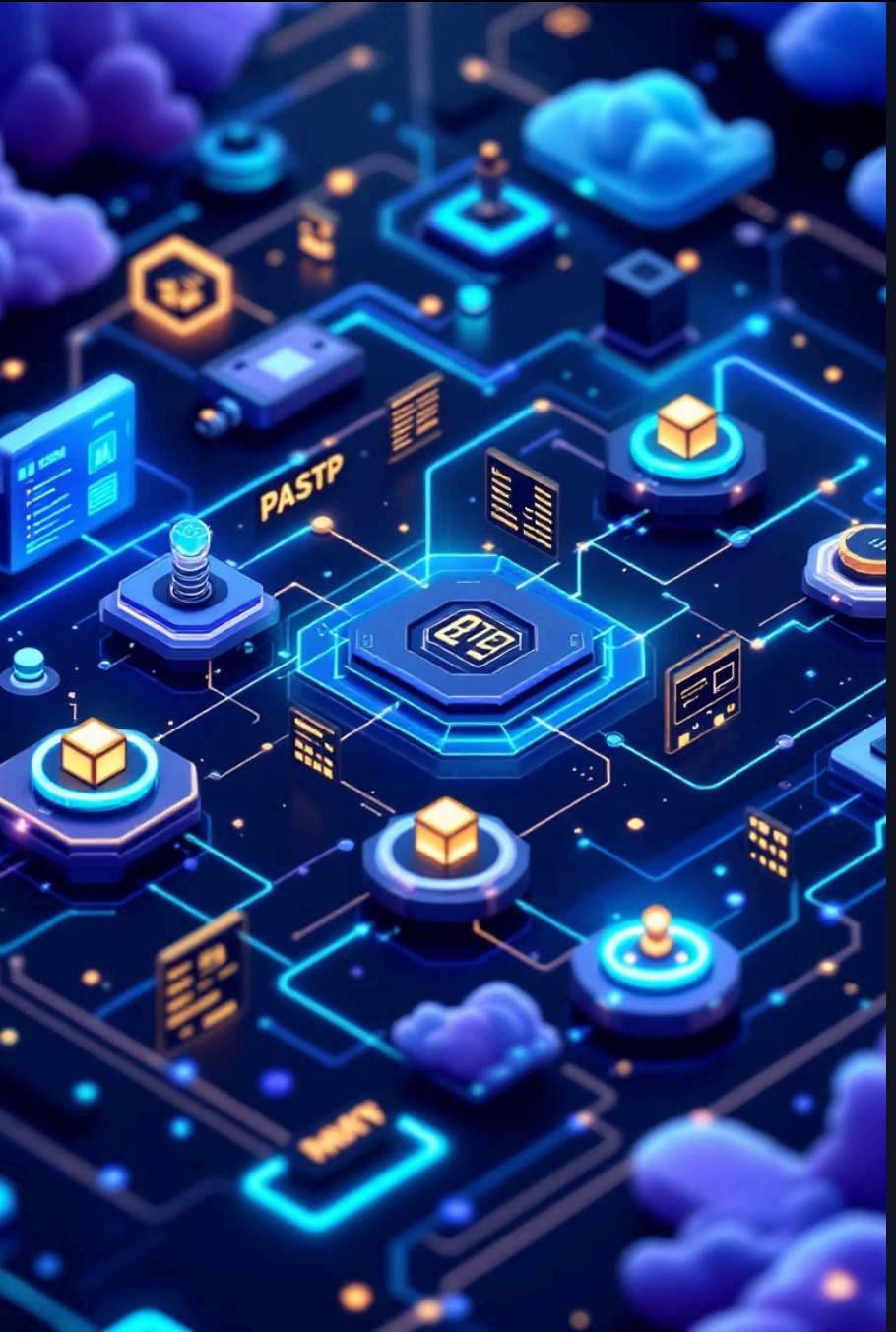
Operações assíncronas não interrompem a execução do código. O programa continua processando outras tarefas enquanto aguarda a conclusão de operações demoradas.

Event Loop

O mecanismo que gerencia eventos e callbacks de forma assíncrona. Quando uma operação é concluída, seu callback é adicionado à fila para execução.

Vantagens

Permite aplicações mais responsivas e eficientes, especialmente em operações de entrada e saída, requisições de rede e processamento de arquivos.



Async/Await e Promises

Sintaxe Async/Await

Simplifica o gerenciamento de operações assíncronas, tornando o código mais legível:

```
const fs = require('fs').promises;

async function lerArquivo() {
  try {
    const data = await
      fs.readFile('arquivo.txt',
                  'utf8');
    console.log(data);
  } catch (err) {
    console.error(err);
  }
}
```

Promises

Representam valores de operações que serão concluídas no futuro. Permitem encadear operações assíncronas de forma organizada.

As Promises possuem três estados: pendente (pending), resolvida (fulfilled) ou rejeitada (rejected), facilitando o controle do fluxo assíncrono.

Pontos Principais

Laços de Repetição

While, for e do...while oferecem diferentes formas de controlar iterações. Escolha a estrutura adequada conforme a necessidade de verificação da condição.

Funções

Declarações de função, expressões e funções de seta permitem organizar código reutilizável com diferentes sintaxes e comportamentos.

Programação Assíncrona

Async/await e Promises tornam o código assíncrono mais legível e gerenciável, aproveitando a natureza não bloqueante do JavaScript.