Lógica de Programação com JavaScript

Sejam bem-vindos! Nesta aula, vamos revisar os fundamentos da lógica de programação usando JavaScript. Desvendaremos os conceitos básicos que formam a base para criar programas incríveis.

O que é Node.js?

Ambiente de Execução

Node.js é uma plataforma de código aberto que permite executar JavaScript fora do navegador web.

Linguagem Versátil

Utilizado para construir aplicações web, APIs, ferramentas de linha de comando e muito mais.

Desenvolvimento de Servidores

Node.js é ideal para criar servidores web eficientes e escaláveis, com foco em aplicações em tempo real.

Introdução à Lógica de Programação

1

O que é Lógica de Programação?

A lógica de programação é a base para a criação de programas de computador. É a arte de pensar de forma estruturada e lógica para resolver problemas, usando instruções passo a passo que um computador pode entender.

2

Por que é importante aprender?

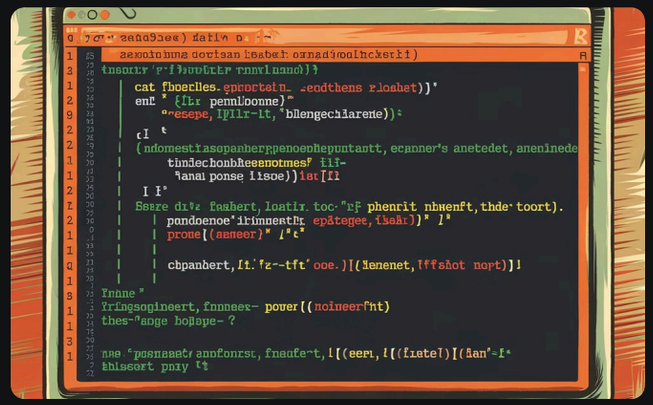
A lógica de programação é uma habilidade valiosa em um mundo cada vez mais digital. Ela desenvolve o pensamento crítico, a criatividade e a capacidade de resolver problemas de forma eficiente. Além disso, abre portas para diversas áreas, como desenvolvimento web, ciência de dados e automação.

3

O que aprenderemos nesta aula?

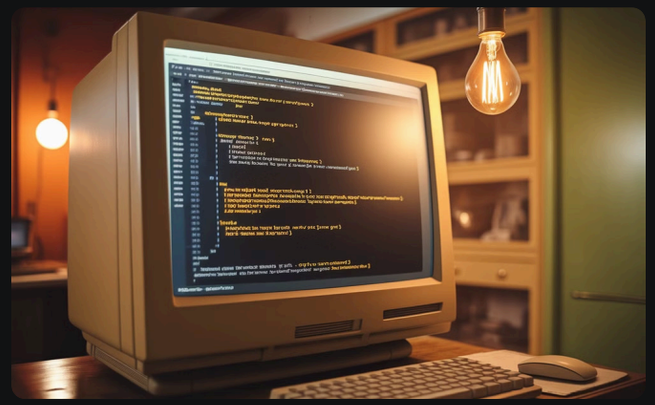
Nesta aula, você aprenderá os fundamentos da lógica de programação, usando a linguagem JavaScript. Você aprenderá sobre tipos de dados, condicionais, repetições, funções e listas, entre outros conceitos importantes.

Tipos de Dados em JavaScript



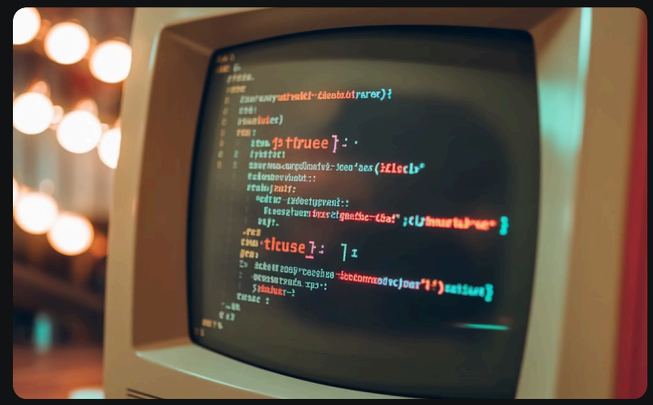
Números

Números são usados para representar valores numéricos, como idade, altura e preços. Eles podem ser inteiros (ex: 10) ou decimais (ex: 3.14).



Strings

Strings são usados para representar texto, como nomes, endereços e mensagens. Eles são delimitados por aspas simples ou duplas (ex: 'Olá, mundo!' ou "JavaScript é legal").



Booleanos

Booleanos representam valores de verdade ou falsidade, como verdadeiro ou falso. Eles são usados para condições e comparações (ex: true ou false).

```
let nome = "Amanda"
let nome1 = 'Amanda'
let numero = 10
let booleano = true
```

JavaScript: Linguagem Tipada

Tipagem Dinâmica

Não precisamos declarar seu tipo. Mas isso só é possível porque ele mesmo identifica o tipo da variável que estamos declarando e o atribui, mesmo que não explicitamente. O operador 'typeof' pode ser usado para exemplificar

```
console.log('Conversão de tipos!');

console.log('ano ' + 2024);
console.log('2' + '2');
//conversão explícita
console.log(parseInt('2') + parseInt('2'));

//conversão implícita
console.log('10' / '2');

console.log('6.5');
console.log('6,5');
```

```
let cliente = 'Maria'
let maiorDeIdade = true
let saldo = 1000

typeof cliente //string
// o js reconhece string pelo uso de aspas, simples ou duplas, ou crase

typeof maiorDeIdade //boolean

typeof saldo //number
```

Variáveis `let` e `const`

Em JavaScript, variáveis podem ser declaradas com a palavra-chave `let` ou `const`.

Let

Variáveis declaradas com `let` podem ter seu valor alterado posteriormente.

```
let nome = "Amanda";
nome = "Maria";
  // retorna Maria
```

Const

Variáveis declaradas com `const` são imutáveis, seu valor não pode ser alterado após a inicialização.

```
const idade = 25;
idade = 30; // Erro:
Atribuição a uma variável
constante
```

Operadores Aritméticos

Adição (+)

Soma dois valores.

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

Subtração (-)

Subtrai um valor de outro.

```
let resultado = 10 - 5; // resultado = 5
```

Multiplicação (*)

Multiplica dois valores.

```
let resultado = 10 * 5; // resultado = 50
```

Divisão (/)

Divide um valor por outro.

```
let resultado = 10 / 5; // resultado = 2
```

Operadores Lógicos

Operadores lógicos são usados para combinar condições e determinar o resultado de uma expressão.

E (&&)

Retorna verdadeiro se ambas as expressões forem verdadeiras. Ex: (5 > 3) && (10 < 20) // true

OU (||)

Retorna verdadeiro se pelo menos uma expressão for verdadeira. Ex: (5 > 3) || (10 < 20) // true

NÃO (!)

Inverte o valor da expressão. Ex: !(5 > 3) // false

Estruturas de Condição (if, else)

1

if

O comando **if** é usado para executar um bloco de código se uma condição for verdadeira.

2

else

O comando **else** é usado para executar um bloco de código se uma condição for falsa.

Estruturas de Condição (if, else)

1

Declaração da Variável

Primeiramente, declare uma variável para armazenar um valor, por exemplo, a idade de uma pessoa.

```
let idade = 20;
```

2

Condicional if

Utilize a estrutura if para verificar se a condição é verdadeira, neste caso, se a idade é maior ou igual a 18.

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

3

Bloco else

Caso a condição do if seja falsa, o bloco else será executado, imprimindo uma mensagem diferente.

```
else {
 console.log("Você é menor de idade!");
}
```

Funções

Blocos Reutilizáveis

Funções são como blocos de código reutilizáveis que realizam tarefas específicas. Elas permitem que você organize seu código de forma mais estruturada e eficiente.

Modularização

Funções dividem seu código em partes menores e mais gerenciáveis, facilitando a manutenção e a compreensão do código.

Reutilização

Você pode chamar uma função quantas vezes quiser, sem precisar escrever o mesmo código repetidamente.

Parâmetros e Retorno de Funções

Parâmetros

Os parâmetros são como ingredientes que você fornece a uma função para que ela faça seu trabalho. Eles são valores que você passa para a função quando ela é chamada.

Retorno

O retorno de uma função é o resultado final do seu trabalho. É o valor que a função fornece de volta ao código que a chamou.

Exemplo

A função `soma(a, b)` recebe dois parâmetros, a e `b`, e retorna a soma deles. Ao chamar a função `soma(2, 3)`, ela retorna o valor 5.

Tipos de Funções em JavaScript

1

Funções Declarativas

Definidas com a palavra-chave `function`, nome, parênteses e chaves. São criadas antes de serem chamadas.

2

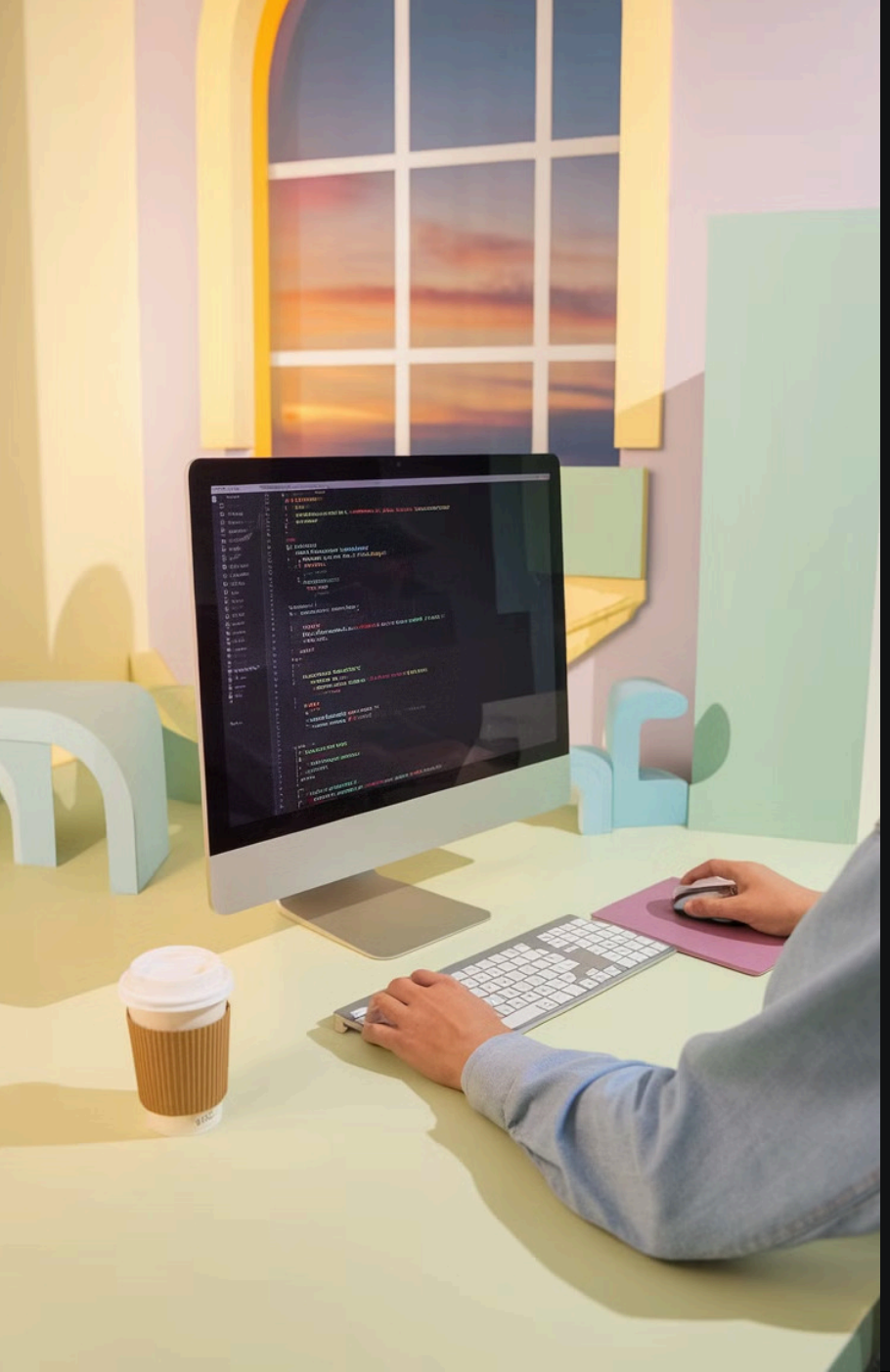
Funções de Expressão

Criadas como variáveis e atribuídas a uma função anônima, usando a palavra-chave `function`. Elas são criadas apenas quando são chamadas.

3

Funções de Seta

Sintaxe mais concisa, com a seta `=>` para definir o corpo da função. São especialmente úteis para funções curtas.



Funções Declarativas em JavaScript

As funções declarativas são definidas usando a palavra-chave `function`, seguida do nome da função, parênteses e chaves.

Elas são criadas antes de serem chamadas, o que significa que você pode chamá-las em qualquer parte do seu código, mesmo antes da sua declaração.

Exemplo

```
function saudacao() {
  console.log("Olá, mundo!");
}
saudacao();
```

Saída

```
Olá, mundo!
```

Funções com Retorno

As funções em JavaScript podem retornar valores, permitindo que você utilize o resultado da função em outras partes do seu código.

Retorno com `return`

A palavra-chave `return` é usada para retornar um valor da função. O código após `return` não é executado.

Uso do Retorno

Você pode usar o valor retornado para realizar cálculos, atribuir a variáveis ou usar em outras funções.

```
// Função que calcula a soma de dois números e retorna o resultado
function soma(a, b) {
    return a + b;
}

// Chamando a função e armazenando o retorno em uma variável
let resultado = soma(5, 3);

// Exibindo o resultado no console
console.log("O resultado da soma é:", resultado);
```

Funções com Parâmetros

Parâmetros são valores que você passa para uma função quando a chama. Eles permitem que você personalize o comportamento da função de acordo com os dados fornecidos.

Dentro da função, os parâmetros são como variáveis locais que armazenam os valores passados. Você pode usar esses parâmetros para realizar cálculos, manipular dados ou executar ações específicas.

```
// Função que exibe uma mensagem personalizada
function saudacao(nome, idade) {
    console.log(`Olá, ${nome}! Você tem ${idade} anos.`);
}

// Chamando a função com diferentes valores de parâmetros
saudacao("Ana", 25);
saudacao("Carlos", 30);
```

Funções de Expressão em JavaScript

1 **Criadas como Variáveis**

Funções de expressão são criadas como variáveis e atribuídas a uma função anônima, usando a palavra-chave \`function\`.

2 **Criadas ao serem Chamadas**

Elas são criadas apenas quando são chamadas, não antes.

3 **Sintaxe**

Variável = function(parâmetro1, parâmetro2) { // código da função }

```
// Definindo uma função de expressão
const saudacao = function(nome, idade) {
    console.log(`Olá, ${nome}! Você tem ${idade} anos.`);
};

// Chamando a função com diferentes valores de parâmetros
saudacao("Ana", 25);
saudacao("Carlos", 30);
```

Funções de Seta em JavaScript

1

Sintaxe Concisa

As funções de seta oferecem uma sintaxe mais compacta em comparação com as funções declarativas e de expressão.

2

Ótimo para Funções Curtas

São especialmente úteis para funções curtas e anônimas, simplificando o código.

```
const estudante = (notaFinal, faltas) => {
    if (notaFinal < 7 && faltas > 4) {
        return true
    }else {
        return false
    }
}
```

```
const exibeNome = (nome) => nome
```

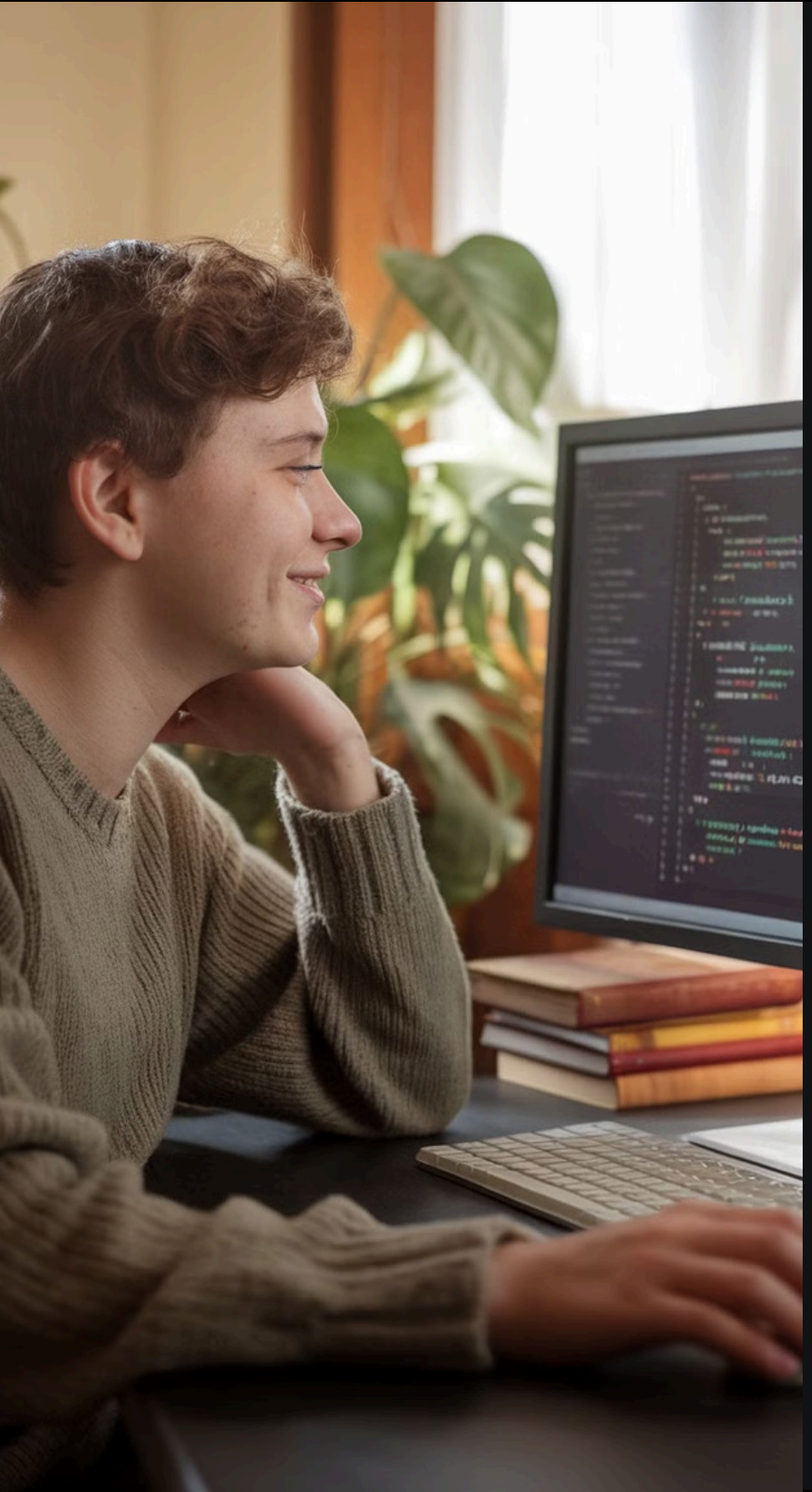
Escopo de Variáveis

Escopo Local

Variáveis declaradas dentro de uma função só podem ser usadas dentro dessa função.

Escopo Global

Variáveis declaradas fora de qualquer função podem ser acessadas por qualquer parte do código.



Exercícios Práticos

1. Declare três variáveis diferentes (uma para cada tipo: string, número e booleano) e atribua valores a elas. Em seguida, exiba o tipo de cada variável no console.
2. Escreva um código que calcule a média de três notas e exiba a mensagem "Aprovado" se a média for maior ou igual a 7, ou "Reprovado" caso contrário.
3. Declare uma variável booleana que informa se está chovendo e utilize-a em uma estrutura condicional (if) para exibir uma mensagem informando se é preciso levar um guarda-chuva ou não dependendo do valor da variável.
4. Crie uma função que recebe um número como parâmetro e retorna o dobro desse número.
5. Crie uma função que receba a idade de uma pessoa e retorne se ela é maior de idade (idade >= 18). Imprima o resultado no console.
6. Crie uma função que receba o nome de uma pessoa como argumento e retorne uma saudação personalizada. Em seguida, chame a função e exiba a saudação no console.