



BACK-END COM JAVASCRIPT

Diego de Souza Rodrigues

JAVASCRIPT BÁSICO INICIANTE

Diego de Souza Rodrigues

conteúdo

- 1**
- 2**
- 3**
- 4**
- 5**
- 6**
- 7**
- 8**
- 9**
- 10**
- 11**
- 12**

Preparando o ambiente de desenvolvimento
Introdução ao javascript
Variaveis e constantes;
Tipos de dados e operadores;
Interações básicas com o usuário com javascript;
Mais sobre strings;
Mais sobre números;
Objeto Math;
Arrays;
Const em Javascript: Conceitos e uso;
Funções em javascript: uma introdução ao básico
Objetos(básico)

Objetivos

Apresentar os conceitos e fundamentos do JavaScript, destacando sua importância no desenvolvimento web e proporcionando uma visão introdutória sobre como a linguagem é utilizada para criar interatividade e dinamismo em páginas e aplicações.

```
ay/hellogit$ git log -10  
* 6a2f33c (HEAD@{10}) HEAD@{10}: checkout: moving from master to novo-recurso  
* 6a2f33c (HEAD@{9}) HEAD@{9}: commit (amend): alterando senha de usuário  
* 6a2f33c (HEAD@{8}) HEAD@{8}: commit (amend): alterando senha de usuário  
* 6a2f33c (HEAD@{7}) HEAD@{7}: commit (amend): alterando senha de usuário  
* 6a2f33c (HEAD@{6}) HEAD@{6}: reset: moving to HEAD@{5}  
* 6a2f33c (HEAD@{5}) HEAD@{5}: commit (amend): alterando senha de usuário  
* 6a2f33c (HEAD@{4}) HEAD@{4}: reset: moving to HEAD@{3}  
* 6a2f33c (HEAD@{3}) HEAD@{3}: commit (amend): alterando senha de usuário  
* 6a2f33c (HEAD@{2}) HEAD@{2}: reset: moving to HEAD@{1}  
* 6a2f33c (HEAD@{1}) HEAD@{1}: commit (amend): alterando senha de usuário  
* 6a2f33c (HEAD) HEAD: commit (amend): alterando senha de usuário  
* 6a2f33c (novo-recurso) HEAD@{0}: reset: moving to HEAD@{-1}
```

JS

Preparando o ambiente de desenvolvimento

```
if (typeof window.innerWidth != 'undefined') {
    wW = window.innerWidth;
} else if (document.documentElement.clientWidth) {
    wW = document.documentElement.clientWidth;
} else if (document.body && document.body.clientWidth) {
    wW = document.body.clientWidth;
}
if (!wH = document.documentElement.scrollHeight) {
    var wh = window.innerHeight || docu
    var wh = !document.all && (scr
}
}
```

NodeJS

VSCode

Google Chrome

Instalando o node

Vamos primeiro instalar o nodeJS

Download em - <https://nodejs.org/pt/download>

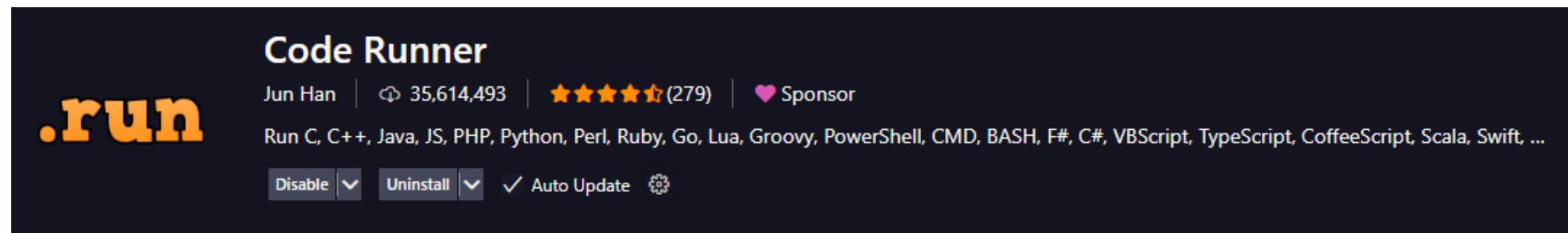


Instalando o VSCode

Download em - <https://code.visualstudio.com/download>

Abra o programa e vamos instalar o plugin chamado Code Runner

- Code runner → executa várias linguagens de programação incluindo o javascript. Fará a ponte entre o javascript executado no vs code e o nodeJS



Instalando o Google Chrome

Download em - <https://www.google.pt/intl/pt-PT/chrome/>



Google Chrome

Introdução ao Javascript

```
if (typeof window.innerWidth != 'undefined') {
    wW = window.innerWidth;
} else if (document.documentElement.clientWidth) {
    wW = document.documentElement.clientWidth;
} else if (document.body && document.body.clientWidth) {
    wW = document.body.clientWidth;
}
if (!wH = document.documentElement.scrollHeight) {
    var wh = window.innerHeight || docu
    var wh = !document.all && (scr
}
}
```

Console.log();
Comentários de código
Navegador vs Node

Console.log()

Está é uma função que utilizaremos pelo resto da nossa carreira como desenvolvedores javascript, para imprimir conteúdo na tela ou debugar nosso código.

Dentro dela podemos colocar qualquer coisa: variáveis, funções, objetos.
Para imprimir valores literários, devemos envolve-la com aspas ou crase

Aspas simples → `console.log('Hello Word!')`

Aspas duplas → `console.log("Hello Word!")`

Crase(template string) → `console.log(`Hello Word!`)`

Comentários de código

Os comentários no JavaScript têm como função inserir anotações dentro do código que não são interpretadas nem executadas pelo navegador ou pelo Node.js. Eles servem exclusivamente para ajudar os desenvolvedores a documentar o raciocínio, explicar trechos de lógica, descrever funcionalidades ou deixar lembretes para manutenção futura do código.

// → comentário de uma linha

/* */ → comentário de múltiplas linhas

/** */ → comentário de múltiplas linhas mas com um asterisco * em cada linha de comentário

Navegador vs Node (HTML + Javascript)

No VSCode, podemos utilizar o codeRunner para executar nosso código diretamente no terminal.

Para executar o código no navegador, temos que criar um arquivo html e incorporar o javascript nele ou incluir o código javascript dentro da tag <script>.

Navegador vs Node (HTML + Javascript)

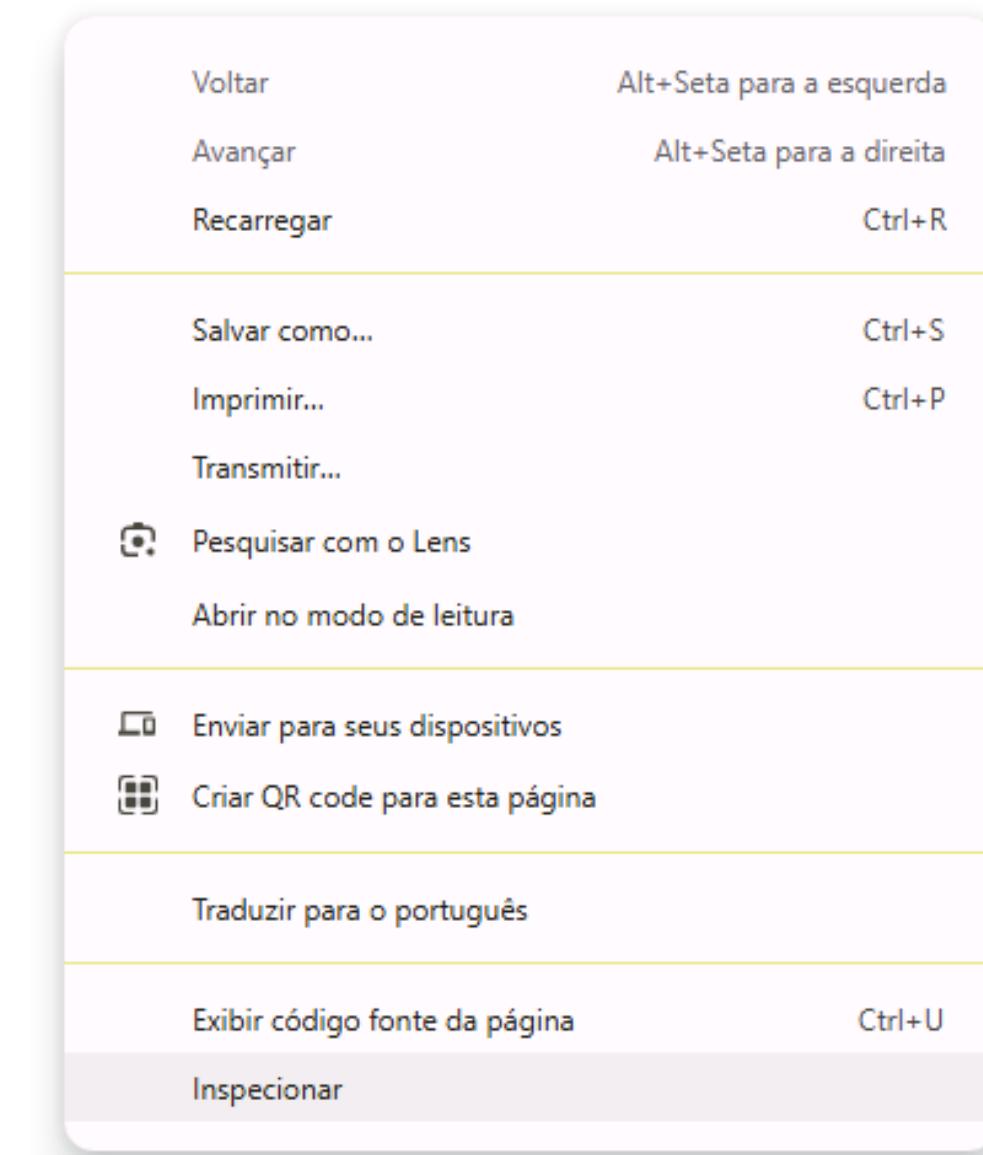
```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Meu primeiro Arquivo html</title>
</head>
<body>
  <script>
    console.log('Olá mundo!')
  </script>
  <script src="index.js"></script>
</body>
</html>
```

Navegador vs Node (HTML + Javascript)

Nosso código será executado automaticamente quando for aberta pelo nosso navegador.

Para ver o código fonte da página podemos digitar **ctrl + u**

Para ver a mensagem passada pelo `console.log()`, precisamos abrir o console do navegador. Pressione **ctrl + shift + i** ou clique com o botão direito mouse na página html e selecione a opção **Inspecionar**



Variáveis e Constantes



let e const

Diferenças entre let, const e var

Exercícios práticos

Variável e constante

Variável é um espaço de memória usado para armazenar dados que podem ser manipulados durante a execução de um programa. Ela funciona como uma “caixa” com um nome (identificador), na qual você guarda um valor que pode ser consultado ou alterado conforme necessário. Em javascript, declaramos variaveis utilizando a palavra reservada **let** ou **var**(legado)

Uma constante é semelhante a uma variável, mas com uma diferença importante: o valor atribuído a ela não pode ser reatribuído. Isso significa que, uma vez declarada, a constante sempre guardará a mesma referência. Em javascript, declaramos constantes utilizando a palavra reservada **const**

let e const

```
let nome = "Diego"; // armazena um texto (string)  
nome = 'Maria'; // agora a variavel terá um novo valor
```

```
const sobrenome = 'Souza'; // não é possível reatribuir um novo sobrenome.  
sobrenome = 'Rodrigues'; // o console retornará um erro
```

let e const

Quando usar?

“Como regra prática, use const sempre que possível e let apenas quando for necessário reatribuir o valor.”

Diferenças entre var, let e const

Característica	var	let	const
Escopo	Função ou global	Bloco { }	Bloco { }
Reatribuição	Permitida	Permitida	Não permitida
Inicialização	pode ser declarada sem valor inicial	pode ser declarada sem valor inicial	Obrigatório declarar com valor inicial
Hoisting	Sofre hoisting (pode ser usada antes da linha de declaração, mas fica undefined)	Sofre hoisting, mas não pode ser usada antes da declaração	Sofre hoisting, mas não pode ser usada antes da declaração
Uso recomendado	Evitar (legado, pode causar erros)	Quando o valor vai mudar	Usar como padrão (quando o valor não muda)

Exercícios

Crie variáveis que armazenem nome, sobrenome, idade, peso e altura em metros. Faça o cálculo do IMC (peso / (alturaEmM * alturaEmM)).

Por fim, imprima as informações seguindo o formato abaixo, mas utilizando os dados de suas variáveis:

***Diego Rodrigues tem 39 anos, pesa 80 kg. Tem 1.72 de altura e seu imc é de
27,041644***

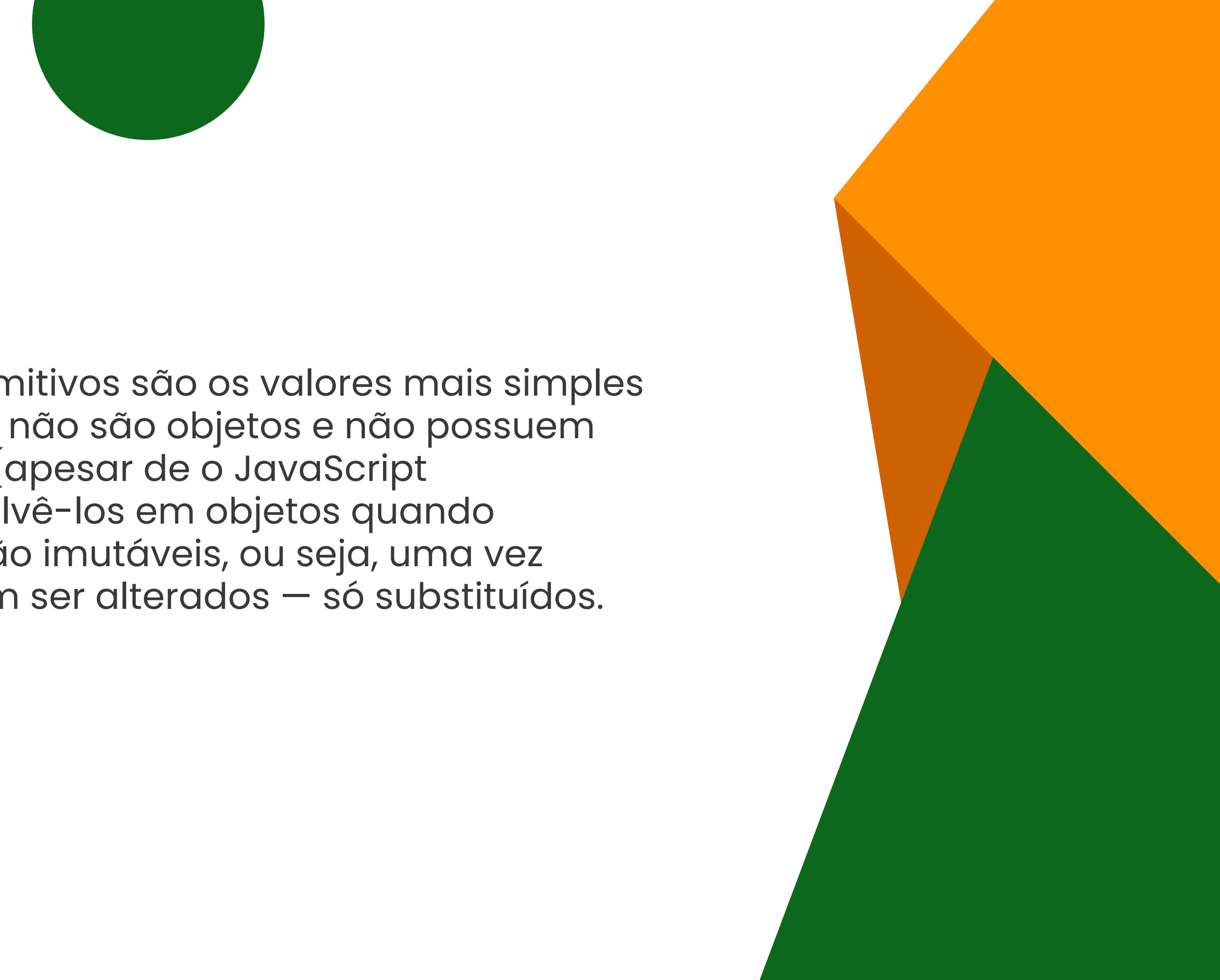
Tipos de dados e operadores



Os tipos de dados

Operadores básicos

Exercícios práticos



Tipos de dados primitivos são os valores mais simples da linguagem, que não são objetos e não possuem métodos próprios (apesar de o JavaScript internamente envolvê-los em objetos quando necessário). Eles são imutáveis, ou seja, uma vez criados, não podem ser alterados — só substituídos.

Tipos de dados primitivos

String – representa textos.

- let nome = "Diego";

Number – representa números inteiros e decimais.

- let idade = 30;
- let altura = 1.75;

Boolean – representa valores lógicos: verdadeiro (true) ou falso (false).

- let ativo = true;

Undefined – quando uma variável é declarada, mas não recebeu valor.

- let x; // undefined

Tipos de dados primitivos

Null – representa ausência intencional de valor.

- `let y = null;`

Symbol – usado para criar identificadores únicos (introduzido no ES6).

- `let id = Symbol("id");`

BigInt – usado para representar números inteiros muito grandes.

- `let big = 123456789012345678901234567890n;`

Operadores básicos

Operador	Função	Quando usar	Exemplo
+	Adição e concatenação	Somar números ou unir strings	$5 + 3 // 8$ "Oi" + "!" // "Oi!"
-	Subtração	Calcular a diferença entre valores	$10 - 4 // 6$
*	Multiplicação	Repetir ou escalar valores numéricos	$6 * 2 // 12$
/	Divisão	Dividir um valor pelo outro	$10 / 2 // 5$
%	Módulo (resto da divisão)	Saber o resto de uma divisão (muito usado em verificações de par/ímpar)	$10 \% 3 // 1$
**	Exponenciação	Elevar um número a uma potência	$2 ** 3 // 8$
++	Incremento	Somar +1 a uma variável	<code>let x = 1; x++; // 2</code>
--	Decremento	Subtrair -1 de uma variável	<code>let y = 5; y--; // 4</code>

Exercício prático

Crie um programa em JavaScript que:

1. Crie as seguintes variáveis:

- nome (string) com o valor de um nome qualquer.
- idade (number) com o valor de sua idade.
- altura (number) com sua altura em metros.
- estudante (boolean) indicando se você é estudante ou não.

2. Utilize operadores aritméticos para calcular:

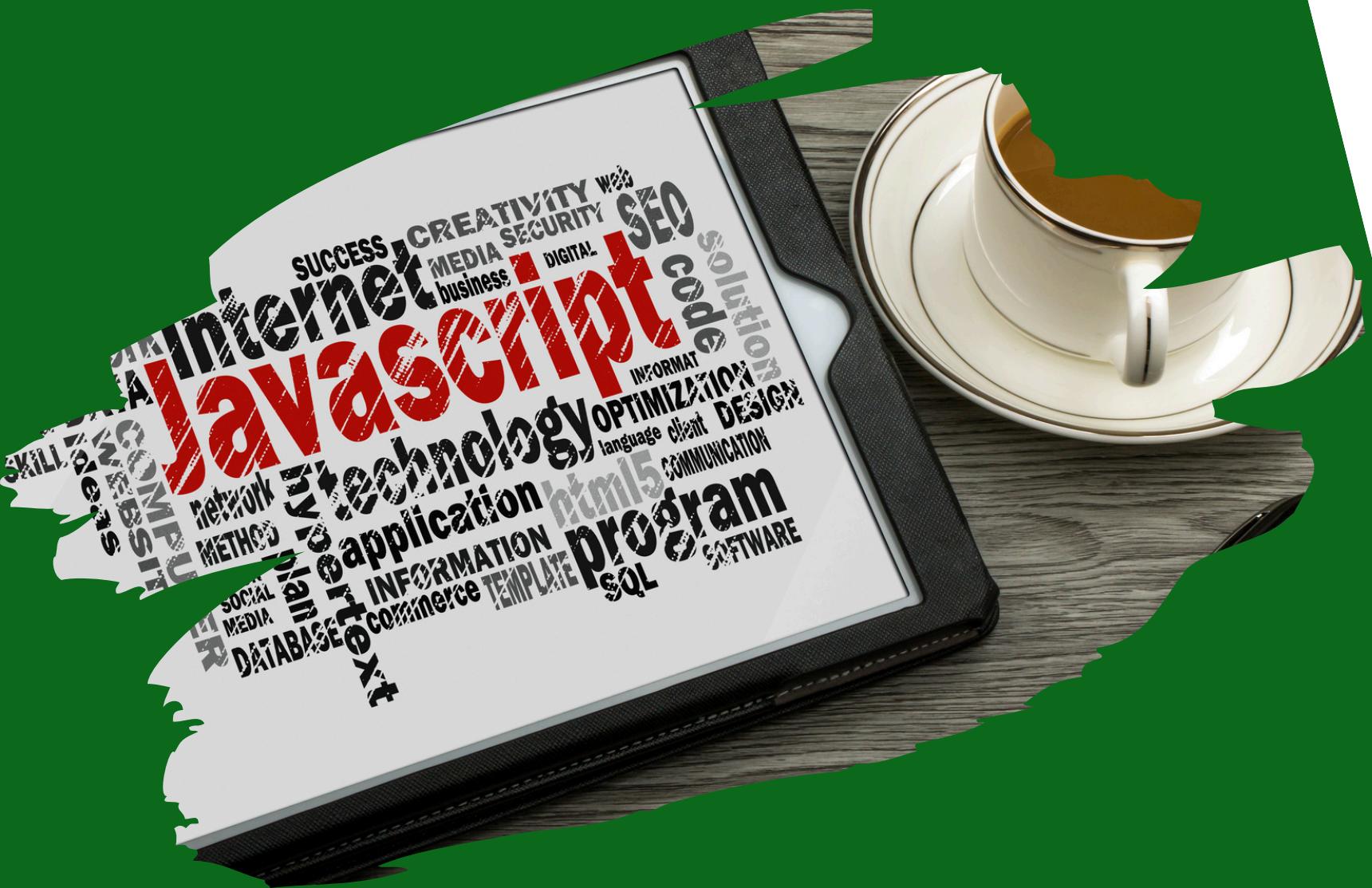
- O dobro da idade.
- A metade da altura.

3. Monte uma frase concatenando as variáveis em uma string no seguinte formato:

4. "Meu nome é NOME, tenho IDADE anos, minha altura é ALTURA metros e sou/ não sou estudante."

5. Mostre todos os resultados no console.

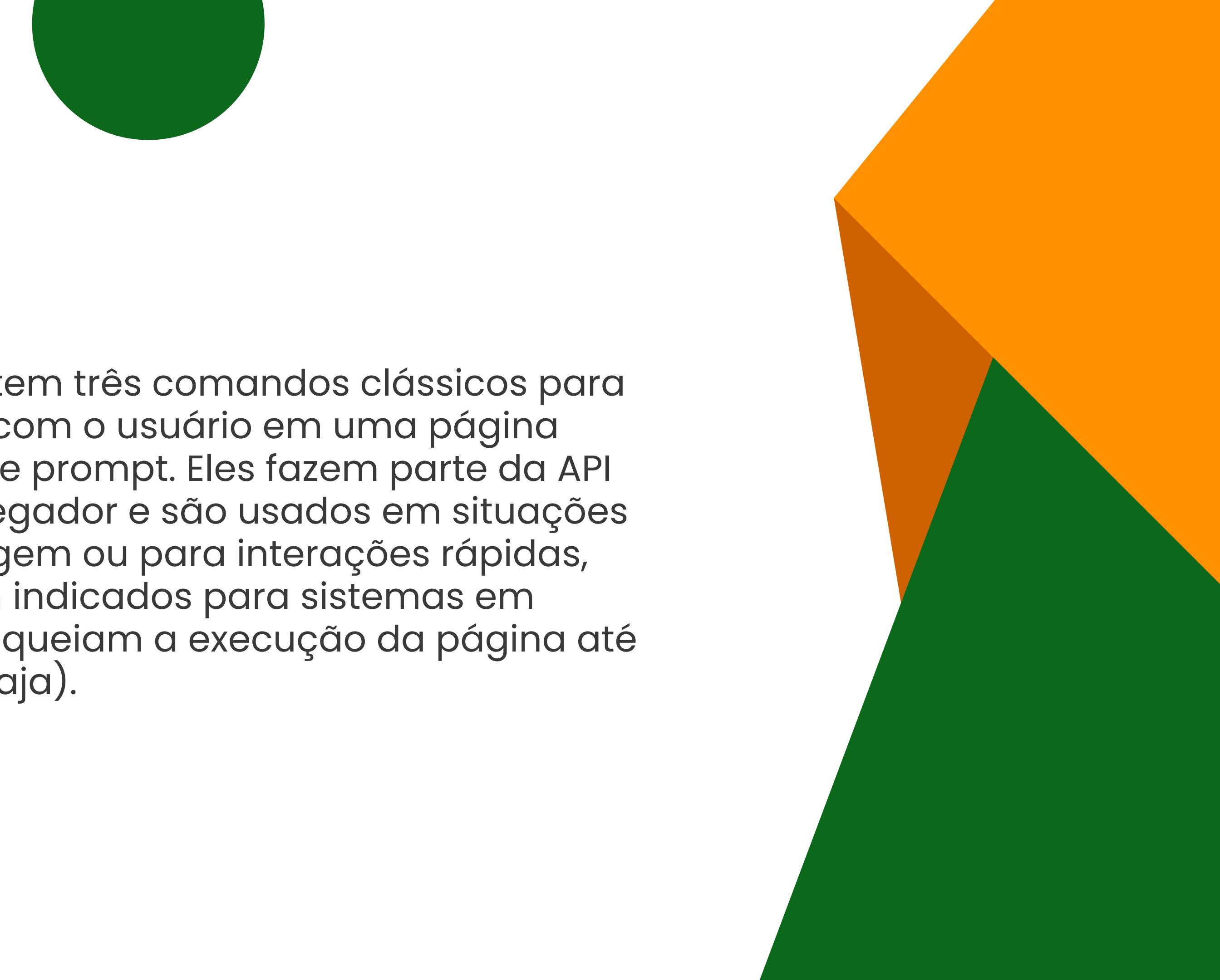
Interações básicas com o usuário em JavaScript



alert

confirm

prompt

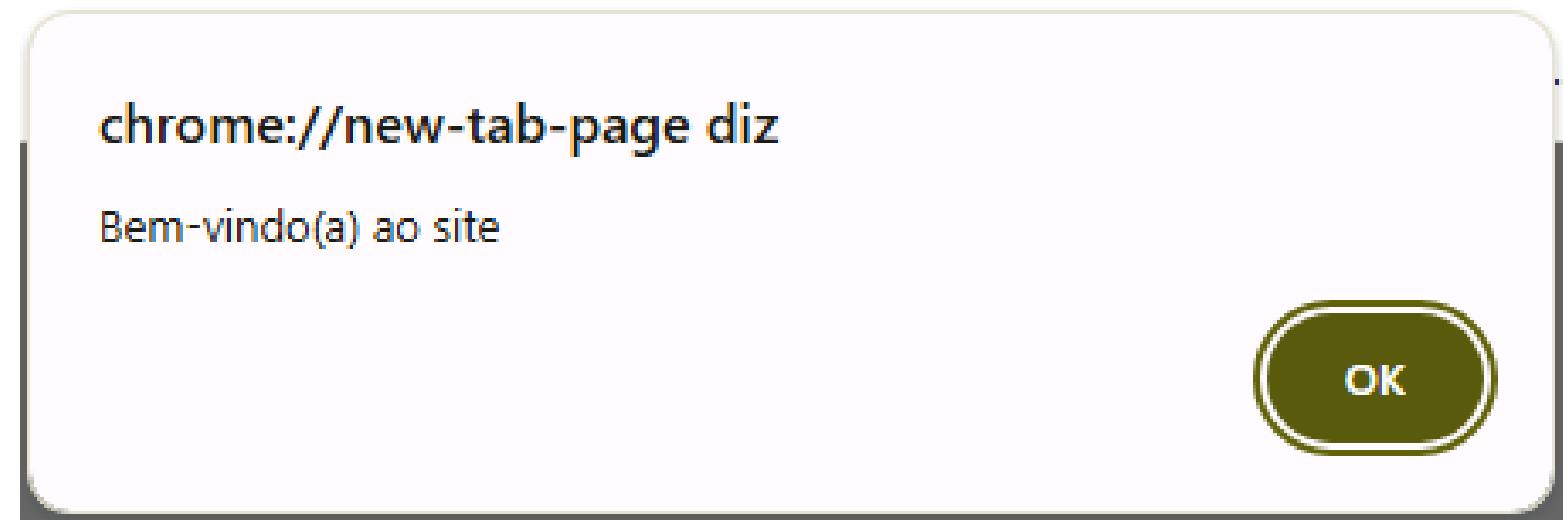


No JavaScript, existem três comandos clássicos para interação simples com o usuário em uma página web: alert, confirm e prompt. Eles fazem parte da API de diálogo do navegador e são usados em situações de teste, prototipagem ou para interações rápidas, embora não sejam indicados para sistemas em produção (pois bloqueiam a execução da página até que o usuário interaja).

alert()

- Descrição: Exibe uma caixa de diálogo com uma mensagem e um botão "OK".
- Uso: Serve para mostrar informações, avisos ou mensagens ao usuário.

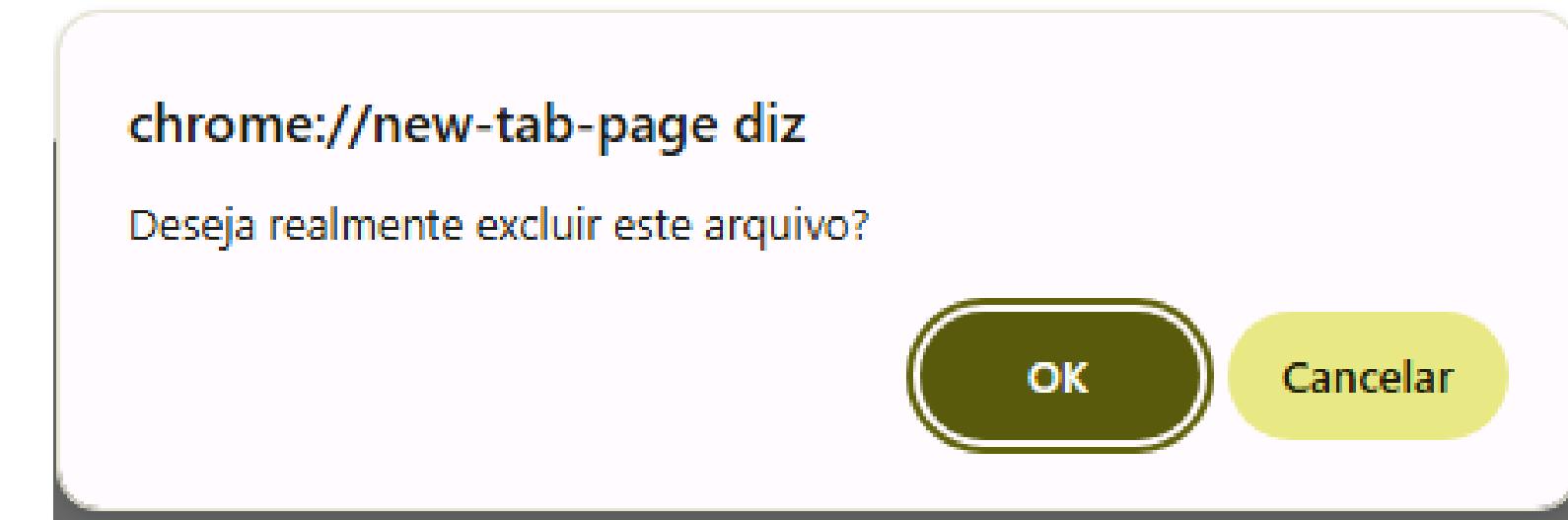
```
alert("Bem-vindo ao site!");
```



confirm()

- Descrição: Exibe uma caixa de diálogo com uma mensagem e dois botões: "OK" e "Cancelar".
- Uso: Permite que o usuário confirme ou cancele uma ação. Retorna true se o usuário clicar em "OK" e false se clicar em "Cancelar".
-

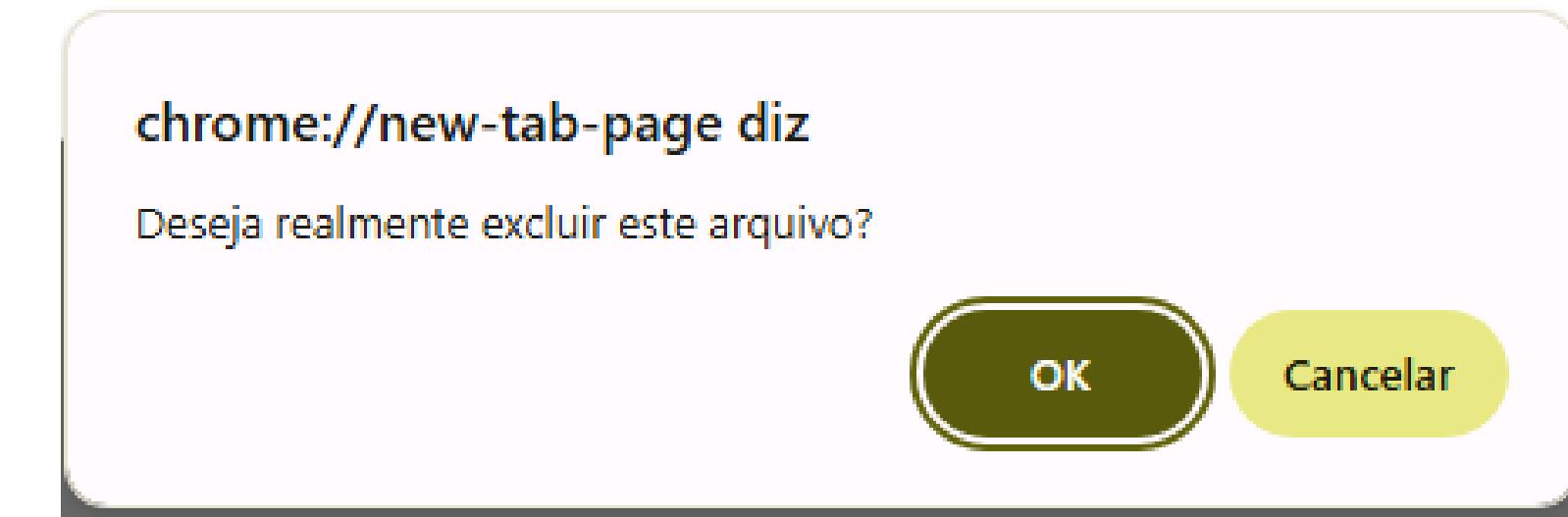
```
let resposta = confirm("Deseja realmente excluir este arquivo?");  
if (resposta) {  
    alert("Arquivo excluído com sucesso!");  
} else {  
    alert("Ação cancelada.");  
}
```



prompt()

- Descrição: Exibe uma caixa de diálogo que solicita ao usuário inserir um valor em um campo de texto.
- Uso: Captura uma entrada do usuário como string. Se o usuário clicar em "Cancelar", o retorno será null.

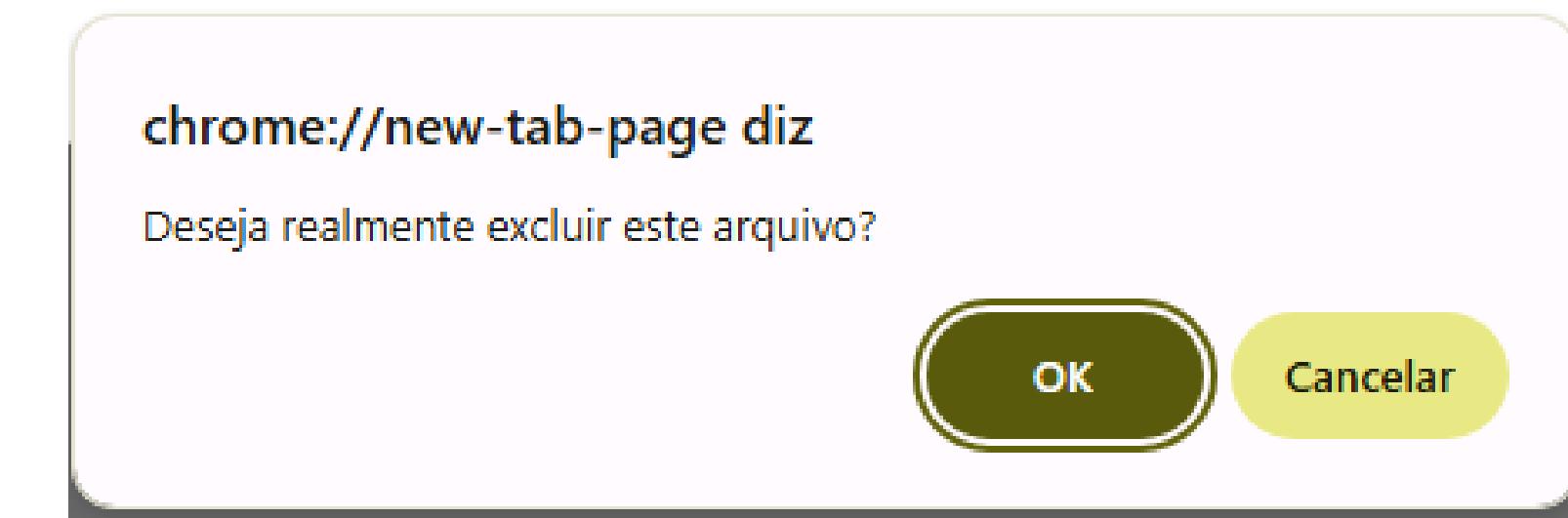
```
let nome = prompt("Qual é o seu nome?");  
if (nome) {  
    alert("Olá, " + nome + "!");  
} else {  
    alert("Você não informou um nome.");  
}
```



prompt()

- Descrição: Exibe uma caixa de diálogo que solicita ao usuário inserir um valor em um campo de texto.
- Uso: Captura uma entrada do usuário como string. Se o usuário clicar em "Cancelar", o retorno será null.

```
let nome = prompt("Qual é o seu nome?");  
if (nome) {  
    alert("Olá, " + nome + "!");  
} else {  
    alert("Você não informou um nome.");  
}
```



Exercício

Você deverá criar uma página web simples que utilize as funções básicas de interação do JavaScript: alert, prompt e confirm.

Etapas:

1. Crie um arquivo chamado index.html.

- Esse arquivo deve conter a estrutura básica de uma página HTML.
- Nele, importe um arquivo JavaScript externo chamado script.js.

2. Crie o arquivo script.js com as seguintes funcionalidades:

- Mostre uma mensagem de boas-vindas para o usuário utilizando alert.
- Pergunte ao usuário seu nome utilizando prompt.
- Pergunte se o nome digitado está correto utilizando confirm.
- Mostre os resultados no console (console.log) e apresente uma mensagem final personalizada com alert, dependendo da resposta do usuário no confirm.

Mais sobre Strings



Aspas dentro de strings

Concatenação

Caracteres de escape

Template Strings (ou Template Literals)

Métodos de String

Aspas dentro de strings

No JavaScript, strings podem ser delimitadas por aspas simples ('), aspas duplas ("") ou crase (`).

Se você usar aspas simples para delimitar a string, pode colocar aspas duplas dentro dela, e vice-versa.

Para usar o mesmo tipo de aspas dentro da string, é preciso usar caracteres de escape.

```
let frase1 = "Ela disse: 'Olá!';  
let frase2 = 'Ele respondeu: "Oi!"';  
let frase3 = "Ele disse: \\"Use aspas duplas\\\"; // com escape
```

Concatenação

Concatenação é o processo de juntar strings. Pode ser feito com o operador + ou +=.

```
let nome = "Diego";
let saudacao = "Olá, " + nome + "!"; // concatenação simples
let mensagem = "Bem-vindo ";
mensagem += "ao JavaScript"; // concatenação com +=
```

Caracteres de escape

Alguns caracteres especiais precisam de uma barra invertida (\) para serem usados dentro da string.

- \" → aspas duplas
- \' → aspas simples
- \\ → barra invertida
- \n → nova linha
- \t → tabulação

```
let texto = "Linha 1\nLinha 2\n\tCom tabulação";
console.log(texto);
```

Template Strings (ou Template Literals)

Usando crases (`), é possível criar strings mais flexíveis:

- Suportam múltiplas linhas sem \n.
- Permitem interpolação de variáveis com \${ }.

```
let nome = "Diego";
```

```
let idade = 30;
```

```
let mensagem = `Olá, meu nome é ${nome}.
```

```
Tenho ${idade} anos de idade.;
```

```
console.log(mensagem);
```

Template Strings (ou Template Literals)

Usando crases (`), é possível criar strings mais flexíveis:

- Suportam múltiplas linhas sem \n.
- Permitem interpolação de variáveis com \${ }.

```
let nome = "Diego";
```

```
let idade = 30;
```

```
let mensagem = `Olá, meu nome é ${nome}.
```

```
Tenho ${idade} anos de idade.;
```

```
console.log(mensagem);
```

Métodos de String

Os métodos de string são funções pré-definidas que pertencem ao objeto String e permitem manipular, analisar ou transformar textos.

Ou seja, uma string em JavaScript é tratada como um objeto imutável (não pode ser alterada diretamente), mas possui diversos métodos que retornam novas strings ou informações sobre ela.

Para exemplificar, utilizarei como exemplo, a declaração abaixo:

```
let campus = "Ouro Preto";
```

Métodos de String

- campus.length → Retorna o número de caracteres da string.
- campus.toUpperCase() → converte todos os caracteres da string para maiúsculas
- campus.toLowerCase() → converte todos os caracteres para minúsculas.
- campus.trim() → remove espaços em branco no início e no fim da string.
- campus.substring(substring) → verifica se a string contém um determinado trecho, retornando true ou false
- campus.indexOf(substring) → retorna o índice da primeira ocorrência de um trecho na string. Se não encontrar, retorna -1.
- campus.lastIndexOf(substring) → retorna o índice da última ocorrência de um trecho.
- campus.slice(início, fim) → extrai parte da string entre os índices informados

Métodos de String

- campus.substring(início, fim) → similar ao slice, mas não aceita índices negativos
- campus.replace(trecho, novoTrecho) → substitui a primeira ocorrência de um trecho por outro
- campus.replaceAll(trecho, novoTrecho) → substitui todas as ocorrências de um trecho.
- campus.split(separador) → divide a string em um array, usando o separador informado.
- campus.startsWith(valor) → Verifica se a string começa com o valor indicado.
- campus.endsWith(valor) → verifica se a string termina com o valor indicado
- campus.repeat(n) → repete a string o número de vezes informado
- campus.concat(string2, string3, ...) → Concatena string. Similar ao operador +

Para conhecer melhor os métodos, acesse:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Numbers_and_strings

Métodos de String

- 1.Crie um arquivo index.html e importe um arquivo JavaScript externo chamado script.js.
- 2.No arquivo script.js, faça o seguinte:
 - a. Crie uma string chamada frase que contenha aspas dentro da string.
 - i.Por exemplo: Ele disse: "JavaScript é incrível!"
 - b. Crie duas variáveis nome e cidade com valores à sua escolha e concatene-as em uma frase completa usando o operador +.
 - c. Crie uma string que utilize caracteres de escape (\n, \t) para formatar uma mensagem de duas linhas com tabulação.
 - d.Crie uma string usando Template String para incluir variáveis e expressões dentro da string de forma dinâmica.
 - i.Exemplo: "Olá, meu nome é NOME e moro em CIDADE."
 - e. Aplique pelo menos três métodos de string em qualquer uma das variáveis criadas:
 - i.toUpperCase(), toLowerCase(), length, replace(), includes(), etc.
- 3.Mostre todas as strings e resultados no console.

Mais sobre números



Tipo único

Valores especiais

Operadores aritméticos

Precisão limitada (ponto flutuante)

Métodos de Number

Funções úteis da classe Number.

Tipo único de número

Em JavaScript, todos os números (inteiros e decimais) são do tipo Number.
Internamente, eles seguem o padrão IEEE 754 (64 bits, ponto flutuante).

```
let inteiro = 10;  
let decimal = 3.14;  
console.log(typeof inteiro); // "number"  
console.log(typeof decimal); // "number"
```

Valores especiais

- Infinity → resultado de divisões por zero.
- -Infinity → infinito negativo.
- NaN (Not-a-Number) → representa uma operação matemática inválida.

```
console.log(1 / 0);    // Infinity  
console.log(-1 / 0);   // -Infinity  
console.log("abc" * 3); // NaN
```

Operadores Aritméticos

Operador	Função	Quando usar	Exemplo
+	Adição e concatenação	Somar números ou unir strings	$5 + 3 // 8$ "Oi" + "!" // "Oi!"
-	Subtração	Calcular a diferença entre valores	$10 - 4 // 6$
*	Multiplicação	Repetir ou escalar valores numéricos	$6 * 2 // 12$
/	Divisão	Dividir um valor pelo outro	$10 / 2 // 5$
%	Módulo (resto da divisão)	Saber o resto de uma divisão (muito usado em verificações de par/ímpar)	$10 \% 3 // 1$
**	Exponenciação	Elevar um número a uma potência	$2 ** 3 // 8$
++	Incremento	Somar +1 a uma variável	<code>let x = 1; x++; // 2</code>
--	Decremento	Subtrair -1 de uma variável	<code>let y = 5; y--; // 4</code>

Precisão limitada

Operações que utilizem números decimais (ponto flutuante) ou que resultem em números decimais (como divisão) podem apresentar imprecisão em cálculos decimais.

```
console.log(0.1 + 0.2); // 0.3000000000000004
```

Uma solução meio que “gambiarra” seria a seguinte:

```
const total = ((0.1 * 100)+(0.2*100))/100 // retorna 0.3
```

Não é muito funcional.

Então para resolver, basta usar o método .toFixed(n).

```
const total = (0.1 + 0.2).toFixed(2); // 0.30
```

Métodos de Number

Em JavaScript, o tipo Number é usado para representar valores numéricos (inteiros e decimais, seguindo o padrão IEEE 754).

Para facilitar operações com esses valores, a linguagem oferece métodos associados ao tipo Number.

Esses métodos são funções pré-definidas que permitem:

- Formatar a exibição de números.
- Converter números em outros formatos (como texto ou bases numéricas).
- Verificar propriedades de valores numéricos (se são inteiros, finitos ou válidos).

Métodos de Number

Métodos de instância

São aplicados em um valor numérico específico (isto é, em variáveis que armazenam números).

Exemplos:

```
let n = 123.456;  
console.log(n.toFixed(2));    // "123.46" → fixa 2 casas decimais  
console.log(n.toPrecision(4)); // "123.5" → define 4 dígitos significativos  
console.log(n.toString(16)); // "7b"   → converte para hexadecimal
```

Métodos de Number

Métodos de instância

São métodos que precisam ser chamados a partir de um valor específico (uma instância do tipo). Ou seja, você usa o método sobre uma variável que contém o dado. No caso de Number, você aplica diretamente sobre um número.

- `numero.toFixed(n)` → Formata o número com n casas decimais, arredondando se necessário.
- `numero.toPrecision(n)` → formata o número para ter n dígitos significativos (antes e depois do separador decimal).
- `numero.toExponential(n)` → retorna o número em notação científica, com n casas decimais
- `numero.toString(base)` → converte o número para string, podendo escolher a base numérica (2 = binário, 16 = hexadecimal, etc);

Métodos de Number

Métodos estáticos (usados direto em Number)

São métodos que pertencem à classe/objeto Number em si, e não às suas instâncias.

Isso significa que você chama diretamente pelo nome do tipo, sem precisar de uma variável.

- `Number.isInteger(n)` → verifica se n é um número inteiro.
- `Number.isFinite(n)` → verifica se n é um número finito (não Infinity, -Infinity ou NaN).
- `Number.isNaN(n)` → verifica se x é NaN (Not-a-Number).
- `Number.parseInt(string, base)` → converte uma string em número interio, podendo especificar a base..
- `Number.parseFloat(string)` → converte uma string em número decimal (ponto flutuante).

Métodos de Number

Constantes do objeto Number

O objeto Number também possui valores especiais já definidos, como:

- `Number.MAX_VALUE` → maior número representável
- `Number.MIN_VALUE` → menor número positivo representável
- `Number.MAX_SAFE_INTEGER` → maior número inteiro que pode ser representado e comparado com segurança, ou seja, sem perda de precisão. (9007199254740991)
- `Number.NaN` → representa um valor numérico indefinido ou irrepresentável, como o resultado de uma operação matemática falhada (por exemplo, a raiz quadrada de um número negativo) ou uma tentativa de converter uma string não numérica para um número. É importante notar que NaN é o único valor em JavaScript que não é igual a si mesmo (`NaN === NaN` retorna false) e o seu tipo é `number`, apesar de o nome sugerir o contrário

Métodos de Number

Comparação rápida

Tipo de método	Como usar	Exemplo
Instância	Chama sobre um valor(variável)	num.toFixed(2)
Estático	Chama direto em Number	Number.isInteger(10)

Exercícios

- 1.Crie um arquivo index.html e importe um arquivo JavaScript externo chamado script.js.
- 2.No arquivo script.js, faça o seguinte:

Parte 1 – Tipos e valores especiais

- Crie uma variável numero1 com valor 10 e numero2 com valor 0.
- Mostre no console:
 - O tipo das variáveis (typeof)
 - O resultado da divisão numero1 / numero2
 - O resultado de Math.sqrt(-1)

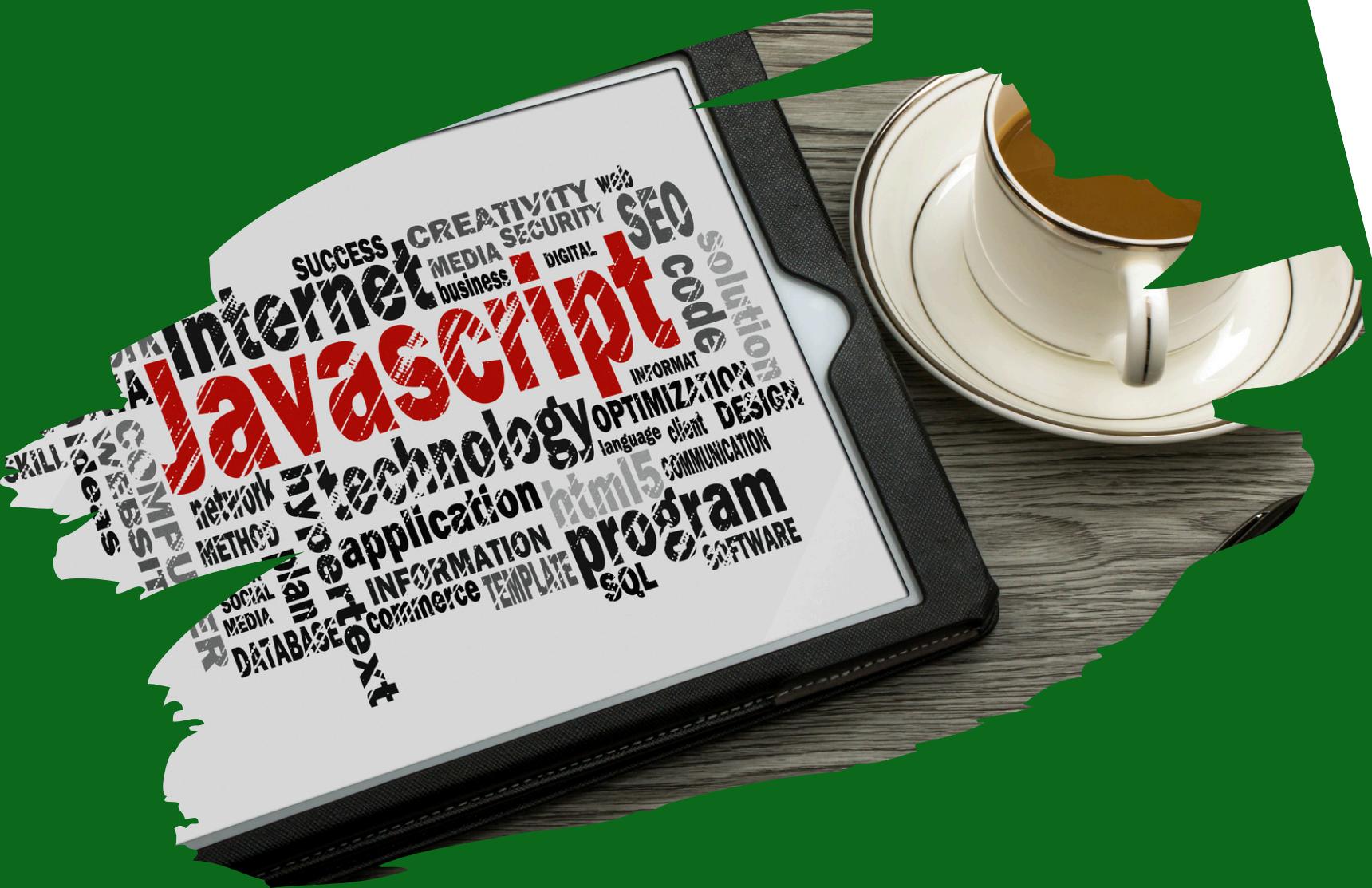
Parte 2 – Operadores e precisão limitada

- Crie duas variáveis a = 0.1 e b = 0.2.
- Some a + b e mostre o resultado no console.
- Mostre o resultado corrigido para duas casas decimais usando métodos de Number.

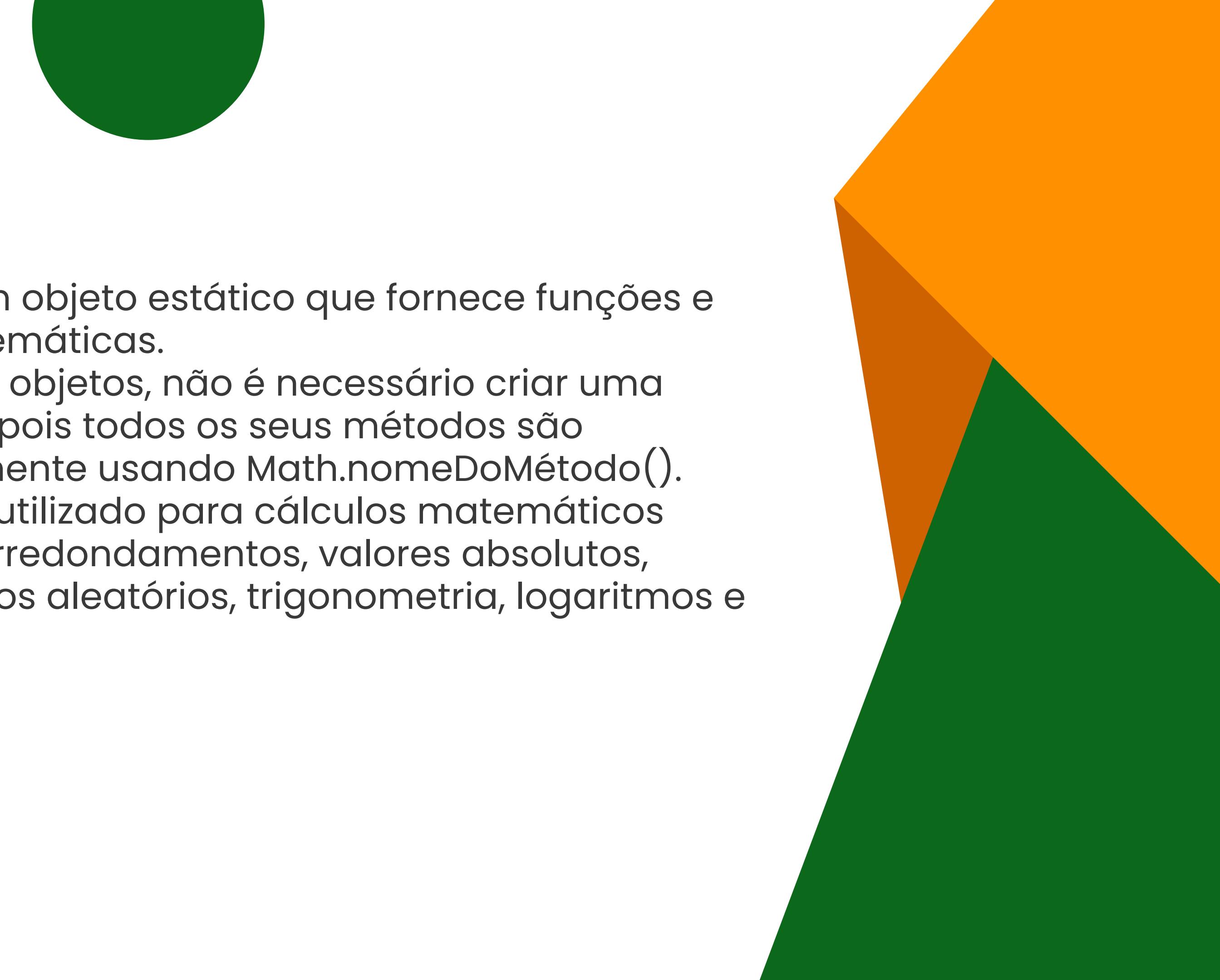
Parte 3 – Métodos e funções úteis da classe Number

- Use os métodos toFixed(), toPrecision(), isNaN() e isFinite() em valores numéricos para demonstrar:
 - Formatação com casas decimais
 - Verificação de número finito
 - Verificação de NaN

Objeto Math



Conceito Métodos



O objeto Math é um objeto estático que fornece funções e propriedades matemáticas.

Diferente de outros objetos, não é necessário criar uma instância de Math, pois todos os seus métodos são chamados diretamente usando `Math.nomeDoMétodo()`.

Ele é amplamente utilizado para cálculos matemáticos como potências, arredondamentos, valores absolutos, geração de números aleatórios, trigonometria, logaritmos e muito mais.

Principais métodos

Suponha a variável

```
let num = 5.35;
```

- `Math.pow(num, 2)` → realiza a exponenciação, elevando um número a uma potência.
- `Math.sqrt(num)` → retorna a raiz quadrada do número.
- `Math.abs(num)` → fornece o valor absoluto, ignorando o sinal.
- `Math.round(num)` → arredonda o valor para o inteiro mais próximo.
- `Math.floor(num)` → arredonda o valor para baixo, para o inteiro mais próximo.
- `Math.ceil(num)` → arredonda o valor para cima, para o inteiro mais próximo.
- `Math.max(a, b, ...)` → retorna o maior valor entre os fornecidos.
- `Math.min(a, b, ...)` → retorna o menor valor entre os fornecidos.

Principais métodos

- `Math.random()` → gera um número pseudoaleatório entre 0 e 1.
- `Math.random() * (10 - 5) + 5` → gera um número aleatorio entre o intervalo definido(5 e 10 onde 10 é o valor máximo e 5 o mínimo)
- `Math.trunc(num)` → remove as casas decimais, ficando apenas com a parte inteira.
- `Math.sign(num)` → retorna o sinal do número (-1, 0 ou 1).
- `Math.log(num)` → calcula o logaritmo natural (base e).
- `Math.log10(num)` → calcula o logaritmo na base 10.
- `Math.exp(num)` → retorna **e** elevado ao número fornecido. Onde **e** é a constante matemática de Euler, aproximadamente 2.71828..., que é a base dos logaritmos naturais.
- `Math.sin(num), Math.cos(num), Math.tan(num)` → funções trigonométricas para seno, cosseno e tangente.

Arrays



O que é um array?

O que é um índice?

Características principais dos arrays

Métodos de Array

O que é um Array?

Um array em JavaScript é uma estrutura de dados usada para armazenar vários valores em uma única variável.

Esses valores podem ser de qualquer tipo: números, strings, objetos ou até outros arrays.

👉 Pense em um array como uma lista ordenada, onde cada item tem uma posição chamada índice.

O que são índices?

Um índice é a posição numérica que identifica cada elemento dentro de uma estrutura de dados, como um array e strings em JavaScript.

Ele funciona como o "endereço" de cada item na lista, permitindo acessar, modificar ou remover valores.

👉 Não esqueça que a contagem de indices de uma estrutura de dados sempre começa com zero. Caso queira começar a considerar o índice do fim para o inicio, o valor inicial é -1

índices (-) -7 -6 -5 -4 -3 -2 -1

índices (+) 0 1 2 3 4 5 6

`const lista = [1, 2, 3, 4, 'oi', 'banana', 'Um texto']`

O que são índices?

Supondo a lista

```
const lista = [1, 2, 5, 10, 'oi', 'banana', ['nome', 'sobrenome', 'idade']];
```

Podemos fazer algumas operações utilizando os índices

- `lista[0]` → acessa o primeiro item do array (1)
- `lista[list.length - 1]` → acessa o último item do array (`['nome', 'sobrenome', 'idade']`)
- `lista.at(-1)` → acessa o último item do array (ES2022);
- `lista.at(-1)[0]` → acessa o último item do array e o primeiro item do respectivo item
- `lista[0] = 10;` → altera o valor do índice zero para 10

Características principais dos arrays

Formas de declarar um array

- Usando colchetes
 - let frutas = ["maçã", "banana", "uva"];
 - let numeros = [1, 2, 3, 4, 5];
- Usando o construtor Array(): cria um novo array usando o objeto Array. Pode ser usado com ou sem elementos iniciais.
 - let cores = new Array("vermelho", "azul", "verde");
 - let vazios = new Array(5); // cria um array com 5 posições vazias
- Array vazio e preenchimento posterior: Cria o array sem elementos e adiciona depois.
 - let numeros = [];
 - numeros.push(10);
 - numeros.push(20);

Métodos de Array

Adicionar ou remover elementos

- `push(elemento)` → adiciona ao final do array.
- `pop()` → remove do final do array.
- `unshift(elemento)` → adiciona ao início do array.
- `shift()` → remove do início do array.
- `splice(início, quantidade, elementos...)` → remove ou adiciona elementos em qualquer posição do array.

Métodos de Array

Adicionar ou remover elementos

- `array.push(elemento)` → adiciona ao final do array.
- `array.pop()` → remove do final do array.
- `array.unshift(elemento)` → adiciona ao início do array.
- `array.shift()` → remove do início do array.
- `array.splice(início, quantidade, elementos...)` → remove ou adiciona elementos em qualquer posição do array.

Métodos de Array

Acessar e localizar elementos

- `array.indexOf(elemento)` → retorna o índice da primeira ocorrência do elemento ou -1 se não existir.
- `array.lastIndexOf(elemento)` → retorna o índice da última ocorrência.
- `array.includes(elemento)` → retorna true se o elemento existir no array, false caso contrário.

Métodos de Array

Acessar e localizar elementos

- `array.indexOf(elemento)` → retorna o índice da primeira ocorrência do elemento ou -1 se não existir.
- `array.lastIndexOf(elemento)` → retorna o índice da última ocorrência.
- `array.includes(elemento)` → retorna true se o elemento existir no array, false caso contrário.

Const em JavaScript: Conceito e Uso



O que é const

Variáveis x Valores

Valores Imutáveis

Valores Mutáveis

O que é const

const descreve uma variável que não pode ser reatribuída (com o operador de atribuição =).
Depois de criá-la, não podemos fazer algo assim:

- `const nome = 'diego';`
- `nome = 'José'; // Erro: Assignment to constant variable.`

Variáveis x Valores

- Variáveis: apelidos ou alias que apontam para valores na memória.
- Valores: dados que realmente ocupam memória.
- Alguns valores são imutáveis:
 - number, string, boolean, undefined, null, symbol, bigint
- Outros valores são mutáveis:
 - arrays, objetos

Valores Imutáveis

- Tipos primitivos são imutáveis
 - `const idade = 25;`
- Uma constante primitiva nunca pode ser alterada, pois tanto a variável quanto o valor são imutáveis.

Valores Mutáveis

- Objetos e arrays são mutáveis, mesmo quando declarados com const.
- A variável continua constante, mas podemos alterar os valores internos:
 - `const array = [1, 2, 3];`
 - `array[0] = 100;` // ✅ permitido
 - `array.push(4);` // ✅ permitido
- A única operação que não funciona é reatribuir a variável:
 - `array = 'Legal';` // ❌ Erro: Assignment to constant variable

Funções em JavaScript: Uma introdução ao básico



- O que é uma função
- Sintaxe básica
- Chamando uma função
- Funções com retorno
- Funções anônimas
- Arrow functions (ES6)

O que é uma função

- Uma função é um bloco de código reutilizável que realiza uma tarefa específica.
- Pode receber entradas (parâmetros) e retornar saídas (valores).
- Funções ajudam a organizar o código, evitando repetição.

Sintaxe básica

```
function nomeDaFuncao(param1, param2) {  
    // código a ser executado  
    return resultado; // opcional  
}
```

- `function` → palavra-chave para declarar uma função
- `nomeDaFuncao` → identificador da função
- `param1, param2` → valores de entrada
- `return` → retorna um valor (opcional)

Funções com retorno

Funções podem retornar valores que podem ser armazenados:

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

```
let resultado = soma(5, 3);  
console.log(resultado); // 8
```

- Toda função pode usar a palavra-chave `return` para devolver um valor.
- Se o `return` não for usado, ou se a função não especificar nenhum valor, o JavaScript retorna `undefined` automaticamente.

Chamando uma função

- Para executar a função, usamos o nome seguido de parênteses:

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

```
saudacao("Diego"); // Olá, Diego!
```

Função anônima

Uma função anônima é uma função sem nome.

Em vez de ser declarada com um identificador, ela geralmente é armazenada em uma variável, passada como parâmetro para outra função, ou usada em eventos e callbacks.

```
const multiplicar = function(a, b) {  
    return a * b;  
};
```

```
console.log(multiplicar(2, 4)); // 8
```

Arrow Function

Uma arrow function (função de seta) é uma sintaxe mais curta para declarar funções em JavaScript, introduzida no ES6.

Ela é especialmente útil para funções anônimas e callbacks.

Características principais

- Sintaxe curta usando =>.
- Não possui seu próprio this, herdando o this do contexto onde foi criada.
- Ideal para funções simples e expressões curtas.
- Pode ser armazenada em variáveis, passada como callback ou usada imediatamente.

Arrow Function

Exemplo1:

```
const soma = (a, b) => a + b;  
console.log(soma(2, 3)); // 5
```

Exemplo 2:

```
const saudacao = (nome) => {  
  const mensagem = "Olá, " + nome + "!";  
  console.log(mensagem);  
};  
saudacao("Diego"); // Olá, Diego!
```

Objetos (Básico)

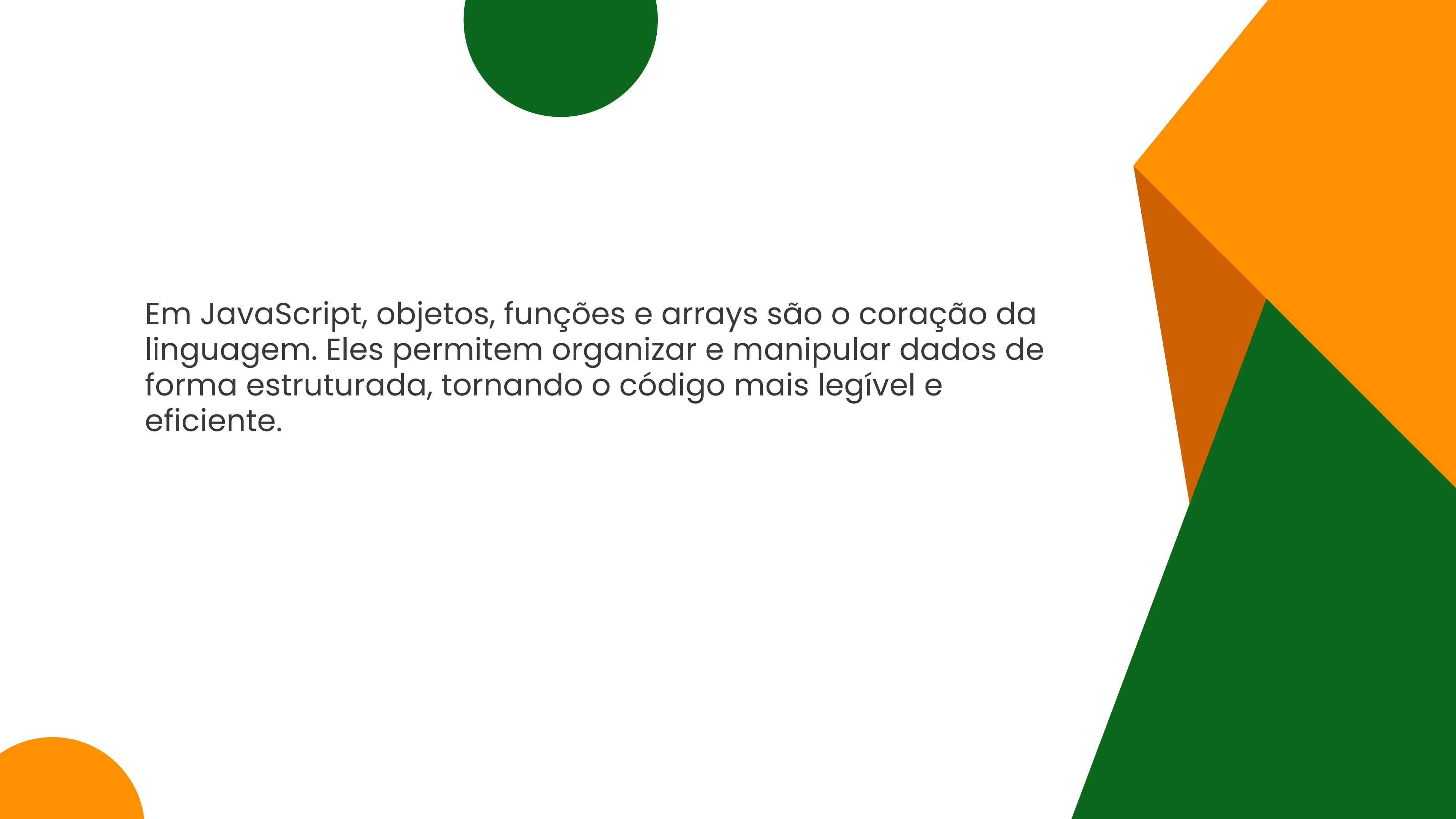


O que é um objeto?

Por que usar objetos?

Métodos e this

Factory Functions



Em JavaScript, objetos, funções e arrays são o coração da linguagem. Eles permitem organizar e manipular dados de forma estruturada, tornando o código mais legível e eficiente.

O que é um objeto em JavaScript?

Um objeto é uma estrutura de dados que permite agrupar vários valores relacionados em uma única entidade.

- Cada valor dentro do objeto é associado a uma chave (key), que funciona como um identificador único.
- Os valores podem ser de qualquer tipo: primitivos (como números e strings), arrays, outros objetos ou funções (quando funções dentro de objetos são chamadas de métodos).

O que é um objeto em JavaScript?

Características importantes

- Agrupa dados relacionados de forma organizada.
- Valores podem ser modificados se forem mutáveis, mesmo se o objeto for declarado com const.
- Pode conter métodos (funções internas) e usar this para referenciar o próprio objeto.

Por que usar objetos?

Imagine que você tem várias variáveis relacionadas, como:

```
const nome01 = 'Diego';
```

```
const sobrenome01 = 'Rodrigues';
```

```
const idade01 = 39;
```

```
const nome02 = 'Maria';
```

```
const sobrenome02 = 'Jose';
```

```
const idade02 = 30;
```

Esse tipo de código fica repetitivo e difícil de manter.

Por que usar objetos?

Com objetos, podemos agrupar dados relacionados em uma única estrutura, sem precisar declarar cada variável separadamente:

```
const pessoa1 = {  
    nome: 'Diego',  
    sobrenome: 'Souza',  
    idade: 39  
};
```

Para acessar os atributos, usamos a notação de ponto:

```
console.log(pessoa1.nome); // Diego
```

Métodos em javascript

- Métodos são funções que pertencem a objetos.
- Eles permitem que o objeto execute ações ou manipule seus próprios dados.
- Diferente de funções comuns, métodos estão associados diretamente a um objeto.

```
const pessoa = {  
    nome: "Diego",  
    idade: 39,  
    // saudacao é um método pertencente ao objeto pessoa  
    saudacao: function() {  
        console.log("Olá, " + this.nome + "!");  
    }  
};  
pessoa.saudacao(); // Olá, Diego!
```

O que é **this**

- **this** é uma palavra-chave especial que dentro de um método refere-se ao próprio objeto.
- Permite acessar ou alterar os atributos do objeto sem precisar repetir o nome do objeto.

O que é **this**

```
const pessoa = {  
    nome: "Diego",  
    idade: 39,  
    aniversario() {  
        this.idade++; // aumenta a idade do próprio objeto  
    },  
    mostraldade() {  
        console.log(this.idade);  
    }  
};
```

```
pessoa.mostraldade(); // 39  
pessoa.aniversario();  
pessoa.mostraldade(); // 40
```

REFERÊNCIAS

- MDN Web Docs. JavaScript: Introdução. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>. Acesso em: 4 set. 2025.
- MDN Web Docs. Variáveis em JavaScript. Disponível em: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Scripting/Variables. Acesso em: 4 set. 2025.
- MDN Web Docs. Tipos de dados e estruturas em JavaScript. Disponível em: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Data_structures. Acesso em: 4 set. 2025.
- MDN Web Docs. Operadores em JavaScript. Disponível em: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_operators. Acesso em: 4 set. 2025.
- MDN Web Docs. Interações com o usuário: alert, prompt e confirm. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/API/Window>. Acesso em: 4 set. 2025.
- MDN Web Docs. Strings em JavaScript. Disponível em: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Scripting/Strings. Acesso em: 4 set. 2025.
- MDN Web Docs. Números em JavaScript. Disponível em: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Numbers_and_strings. Acesso em: 4 set. 2025.
- MDN Web Docs. Objeto Math em JavaScript. Disponível em: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math. Acesso em: 4 set. 2025.
- MDN Web Docs. Arrays em JavaScript. Disponível em: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Scripting/Arrays. Acesso em: 4 set. 2025.
- MDN Web Docs. Const em JavaScript. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const>. Acesso em: 4 set. 2025.
- MDN Web Docs. Funções em JavaScript. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions>. Acesso em: 4 set. 2025.
- MDN Web Docs. Objetos em JavaScript. Disponível em: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects. Acesso em: 4 set. 2025.