# Javascript

[Declaração de Variáveis 2](#_Toc204083030)

[Escopo: 2](#_Toc204083031)

[Reatribuição: 2](#_Toc204083032)

[Redeclaração: 2](#_Toc204083033)

[Exemplos: 2](#_Toc204083034)

[Recomendações: 3](#_Toc204083035)

[Verificação de Tipo de variável 4](#_Toc204083036)

[Verificação de tipos com typeof: 4](#_Toc204083037)

[Verificação de tipos com Object.prototype.toString: 4](#_Toc204083038)

[Caixas de Diálogos 6](#_Toc204083039)

[Caixas de Diálogo em JavaScript 6](#_Toc204083040)

[Considerações Adicionais 6](#_Toc204083041)

[Operadores 7](#_Toc204083042)

[Aritméticos 7](#_Toc204083043)

[Outros Operadores: 7](#_Toc204083044)

[Lógicos 7](#_Toc204083045)

[Comparação 8](#_Toc204083046)

[Controle de Fluxo 10](#_Toc204083047)

[Funções 12](#_Toc204083048)

[Tipos de funções 12](#_Toc204083049)

[Função Declarada 12](#_Toc204083050)

[Função expressa 13](#_Toc204083051)

[Função de seta 13](#_Toc204083052)

[DOM – Document Object Model 14](#_Toc204083053)

[O que é o DOM? 14](#_Toc204083054)

[Relação entre DOM e JavaScript 14](#_Toc204083055)

## Declaração de Variáveis

Em JavaScript, var, let e const são usados para declarar variáveis, mas possuem diferenças importantes em relação ao escopo e à capacidade de reatribuição e redeclaração.

var tem escopo de função e pode ser redeclarado e reatribuído.

let tem escopo de bloco e pode ser reatribuído, mas não redeclarado.

const tem escopo de bloco e não pode ser reatribuído nem redeclarado.

### Escopo:

* var:

Possui escopo de função. Se uma variável var for declarada dentro de uma função, ela só estará disponível dentro dessa função. Se for declarada fora de qualquer função, terá escopo global.

* let e const:

Possuem escopo de bloco. Variáveis declaradas com let e const são acessíveis apenas dentro do bloco onde foram declaradas (um bloco é delimitado por chaves {}).

### Reatribuição:

* var e let:

Permitem a reatribuição de valores, ou seja, você pode alterar o valor de uma variável declarada com var ou let várias vezes.

* const:

Não permite a reatribuição. Uma vez que um valor é atribuído a uma variável const, ele não pode ser alterado.

### Redeclaração:

* var:

Permite a redeclaração, ou seja, você pode declarar uma variável var várias vezes com o mesmo nome no mesmo escopo, e a última declaração prevalecerá.

* let e const:

Não permitem a redeclaração no mesmo escopo. Tentar redeclarar uma variável com let ou const no mesmo escopo resultará em um erro.

### Exemplos:

function exemploVar() {  
 var x = 10;  
 if (true) {  
 var x = 20; // Redeclara e reatribui x  
 console.log(x); // Saída: 20

}  
 console.log(x); // Saída: 20 (x ainda é 20, pois var tem escopo de função)

}

function exemploLet() {  
 let y = 10;  
 if (true) {

let y = 20; // Declara uma nova variável y com escopo de bloco

console.log(y); // Saída: 20

}

console.log(y); // Saída: 10 (y original)

}

function exemploConst() {  
 const z = 10;  
 // z = 20; // Erro: Cannot assign to read only property 'z' of object '#<Object>'

if (true) {  
 // const z = 20; // Erro: Identifier 'z' has already been declared

// console.log(z);

}

// console.log(z);

}

exemploVar();

exemploLet();

exemploConst();

### Recomendações:

* É geralmente recomendado evitar var em favor de let e const, pois let e const oferecem um controle de escopo mais preciso e ajudam a evitar erros comuns.
* Use const para declarar variáveis cujo valor não deve ser alterado após a inicialização.
* Use let para declarar variáveis que podem ter seus valores alterados ao longo do tempo, mas que precisam de escopo de bloco.

## Verificação de Tipo de variável

Em JavaScript, a verificação de tipos é um conceito importante devido à natureza dinamicamente tipada da linguagem. O operador typeof é a ferramenta mais comum para determinar o tipo de uma variável, mas possui peculiaridades, especialmente com valores nulos e arrays. Para verificações mais precisas, métodos adicionais podem ser necessários.

### Verificação de tipos com typeof:

O operador typeof retorna uma string que descreve o tipo da variável. Por exemplo:

typeof "Hello" // "string"  
typeof 123 // "number"  
typeof true // "boolean"  
typeof {} // "object"  
typeof [] // "object" (peculiaridade)  
typeof null // "object" (peculiaridade)  
typeof function(){} // "function"

**Peculiaridades do typeof:**

* Retorna "object" para arrays e valores nulos, o que pode ser enganoso.
* Para arrays, a função Array.isArray() pode ser usada para uma verificação mais precisa:

Array.isArray([]) // true

* Para valores nulos, a comparação com null diretamente é mais recomendada:

const myVar = null;  
if (myVar === null) {  
 // ...  
}

### Verificação de tipos com Object.prototype.toString:

Uma solução mais robusta para verificação de tipos, especialmente para objetos, é usar Object.prototype.toString. Esta função, quando chamada com call ou apply, retorna uma string formatada como "[object Tipo]":

function tipo(obj) {  
 return Object.prototype.toString.call(obj).slice(8, -1);  
}  
  
tipo("Hello") // "String"  
tipo(123) // "Number"  
tipo([]) // "Array"  
tipo({}) // "Object"  
tipo(null) // "Null"  
tipo(undefined) // "Undefined"

**Comparação de igualdade estrita:**

O operador === (igualdade estrita) compara valores e tipos. É recomendado para evitar comportamentos inesperados em comparações:

0 === +0 // true  
0 === -0 // true  
NaN === NaN // false (NaN é sempre diferente de si mesmo)

O operador == (igualdade) tenta converter os operandos antes de comparar, o que pode levar a resultados surpreendentes:

"0" == 0 // true (conversão para número)

null == undefined // true (conversão para booleano)

## Caixas de Diálogos

Em JavaScript, caixas de diálogo (também conhecidas como pop-ups ou caixas modais) são usadas para interagir com o usuário, exibindo mensagens e solicitando informações. Existem três tipos principais de caixas de diálogo: alert, confirm e prompt.

### Caixas de Diálogo em JavaScript

**Alert:** Exibe uma mensagem informativa ao usuário, geralmente para avisos ou notificações.

alert("Esta é uma mensagem de alerta!");

**Confirm:** Solicita ao usuário uma confirmação, retornando true se o usuário clicar em "OK" e false se clicar em "Cancelar".

let resposta = confirm("Tem certeza que deseja continuar?");  
 if (resposta) {  
 // Ação a ser executada se o usuário clicar em OK

} else {

// Ação a ser executada se o usuário clicar em Cancelar  
 }

**Prompt:** Solicita ao usuário que insira um valor, retornando o valor digitado ou null se o usuário clicar em "Cancelar".

let nome = prompt("Digite seu nome:", "Nome");  
 if (nome != null) {  
 // Ação a ser executada com o nome digitado

} else {  
 // Ação a ser executada se o usuário clicar em Cancelar

}

### Considerações Adicionais

* As caixas de diálogo em JavaScript são modais, o que significa que a execução do código é interrompida até que o usuário interaja com a caixa de diálogo.
* A aparência das caixas de diálogo é determinada pelo sistema operacional e configurações do navegador, não sendo customizável com CSS.
* É importante usar as caixas de diálogo com moderação, pois elas podem interromper a experiência do usuário.
* Para interfaces mais customizáveis, considere criar caixas de diálogo usando elementos HTML e manipulação do DOM.

## Operadores

### Aritméticos

Os operadores aritméticos em JavaScript são usados para realizar operações matemáticas em valores numéricos. Os operadores mais comuns são: adição (+), subtração (-), multiplicação (\*), divisão (/), módulo (%), incremento (++) e decremento (--).

Operadores Aritméticos Comuns:

* **Adição (+):** Soma dois valores. Ex: 5 + 3 resulta em 8.
* **Subtração (-):** Subtrai o segundo valor do primeiro. Ex: 10 - 4 resulta em 6.
* Multiplicação ():\*: Multiplica dois valores. Ex: 6 \* 7 resulta em 42.
* **Divisão (/):** Divide o primeiro valor pelo segundo. Ex: 20 / 5 resulta em 4.
* **Módulo (%):** Retorna o resto da divisão. Ex: 10 % 3 resulta em 1.
* **Incremento (++)**: Aumenta o valor de uma variável em 1. Ex: x++ (onde x inicialmente é 5, x será 6 após a operação).
* **Decremento (--):** Diminui o valor de uma variável em 1. Ex: x-- (onde x inicialmente é 5, x será 4 após a operação).

let num1 = 10;  
let num2 = 5;  
  
let soma = num1 + num2; // soma = 15  
let subtracao = num1 - num2; // subtracao = 5  
let multiplicacao = num1 \* num2; // multiplicacao = 50  
let divisao = num1 / num2; // divisao = 2  
let resto = num1 % num2; // resto = 0  
  
let x = 7;  
x++; // x agora é 8

x--; // x agora é 7

### Outros Operadores:

* **Exponenciação (**) :\*\* Eleva um número à potência de outro. Ex: 2 \*\* 3 (2 elevado à 3ª potência) resulta em 8.

### Lógicos

Em JavaScript, operadores lógicos são usados para combinar expressões booleanas e criar condições compostas. Os principais operadores lógicos são: && (AND), || (OR) e ! (NOT).

1. Operador AND (&&):

* Retorna true se ambos os operandos forem true. Caso contrário, retorna false.
* Exemplo: (5 > 3) && (10 < 20) retorna true.
* Exemplo: (5 > 3) && (10 > 20) retorna false.

2. Operador OR (||):

* Retorna true se pelo menos um dos operandos for true. Se ambos forem false, retorna false.
* Exemplo: (5 > 3) || (10 > 20) retorna true.
* Exemplo: (5 < 3) || (10 > 20) retorna false.

3. Operador NOT (!):

* Inverte o valor booleano do operando.
* Se o operando for true, retorna false. Se o operando for false, retorna true.
* Exemplo: !(5 > 3) retorna false.
* Exemplo: !(5 < 3) retorna true.

Precedência:

* A ordem de avaliação de operadores lógicos é: !, &&, ||.
* É recomendado usar parênteses para garantir a ordem desejada e facilitar a leitura do código.

let idade = 25;  
let temCarteira = true;  
  
if (idade >= 18 && temCarteira) {  
 console.log("Pode dirigir");  
} else {  
 console.log("Não pode dirigir");  
}  
  
let maiorDeIdade = idade >= 18;  
if (maiorDeIdade || temCarteira) {  
 console.log("Pode entrar");  
}  
  
let naoPodeEntrar = !(idade >= 18);

### Comparação

Em JavaScript, operadores de comparação são usados para comparar valores e retornar um valor booleano (verdadeiro ou falso) com base no resultado da comparação. Eles são cruciais para a tomada de decisões em estruturas condicionais e loops.

Operadores de Comparação:

* == (Igual a):

Compara se dois valores são iguais após a conversão de tipo. Ex: 5 == "5" retorna true.

* === (Estritamente igual a):

Compara se dois valores são iguais e do mesmo tipo, sem conversão de tipo. Ex: 5 === "5" retorna false.

* != (Diferente de):

Compara se dois valores são diferentes após a conversão de tipo. Ex: 5 != "5" retorna false.

* !== (Estritamente diferente de):

Compara se dois valores são diferentes ou de tipos diferentes, sem conversão de tipo. Ex: 5 !== "5" retorna true.

* > (Maior que):

Compara se o valor da esquerda é maior que o da direita. Ex: 10 > 5 retorna true.

* < (Menor que):

Compara se o valor da esquerda é menor que o da direita. Ex: 5 < 10 retorna true.

* >= (Maior ou igual a):

Compara se o valor da esquerda é maior ou igual ao da direita. Ex: 10 >= 10 retorna true.

* <= (Menor ou igual a):

Compara se o valor da esquerda é menor ou igual ao da direita. Ex: 5 <= 10 retorna true.

let num1 = 10;  
let num2 = "10";  
let num3 = 5;  
  
console.log(num1 == num2); // true (conversão de tipo)

console.log(num1 === num2); // false (tipos diferentes)

console.log(num1 != num3); // true

console.log(num1 !== num2); // true

console.log(num1 > num3); // true

console.log(num3 < num1); // true

console.log(num1 >= num2); // true

console.log(num3 <= num1); // true

**Diferença entre == e ===:**

A principal diferença entre == e === reside na conversão de tipo. O operador == realiza a conversão de tipos antes de fazer a comparação, enquanto o === não. Portanto, o === é mais seguro para evitar erros inesperados, especialmente quando se compara valores de tipos diferentes.

## Controle de Fluxo

Em JavaScript, controle de fluxo refere-se à ordem em que as instruções de um programa são executadas. Sem controle de fluxo, o código é executado sequencialmente, linha por linha, de cima para baixo. Estruturas de controle de fluxo, como condicionais e loops, permitem que o código tome decisões e execute diferentes blocos de código com base em condições específicas ou repita um bloco de código várias vezes.

Estruturas de Controle de Fluxo:

* **Condicionais:**
  + if, else if, else: Permitem executar diferentes blocos de código com base em uma condição booleana.
  + switch: Permite comparar um valor com vários casos e executar o bloco de código correspondente ao caso que corresponder.
* **Loops:**
  + for: Permite executar um bloco de código um número específico de vezes.
  + while: Permite executar um bloco de código enquanto uma condição for verdadeira.
  + do...while: Semelhante ao while, mas garante que o bloco de código seja executado pelo menos uma vez.
  + for...in: Permite iterar sobre as propriedades de um objeto.
  + for...of: Permite iterar sobre os valores de um objeto iterável (como arrays, strings, etc.).
* **Outras estruturas:**
  + break: Sai do loop ou switch atual.
  + continue: Pula para a próxima iteração do loop atual.

// Exemplo de if/else  
let idade = 20;  
  
if (idade >= 18) {  
 console.log("Maior de idade");  
} else {  
 console.log("Menor de idade");  
}  
  
// Exemplo de switch  
let dia = new Date().getDay();  
let nomeDia;  
  
switch (dia) {  
 case 0:  
 nomeDia = "Domingo";  
 break;  
 case 1:  
 nomeDia = "Segunda-feira";  
 break;  
 // ... (outros dias)  
 default:  
 nomeDia = "Dia inválido";  
}  
  
console.log("Hoje é " + nomeDia);  
  
// Exemplo de for

for (let i = 0; i < 5; i++) {  
 console.log(i);  
}  
  
// Exemplo de while

let j = 0;  
while (j < 3) {  
 console.log("Valor de j: " + j);  
 j++;  
}

Em resumo, o controle de fluxo é essencial para criar programas JavaScript dinâmicos e interativos, permitindo que o código execute ações diferentes com base em condições e itere sobre dados de forma eficiente.

## Funções

Em JavaScript, uma função é um bloco de código reutilizável que executa uma tarefa específica. Funções são definidas usando a palavra-chave function, seguida por um nome (opcional para funções anônimas), parênteses () para argumentos, e chaves {} para o corpo da função. Elas podem retornar um valor usando a palavra-chave return.

### Tipos de funções

* **Funções Declaradas:** Definidas com a palavra-chave function seguida pelo nome da função.
* **Funções Expressas:** Atribuídas a uma variável. Podem ser nomeadas ou anônimas.
* **Funções de seta (Arrow functions):** Uma forma mais concisa de escrever funções, especialmente útil para funções simples e aninhadas.

### Função Declarada

function nomeDaFuncao(parametro1, parametro2) {  
 // Corpo da função

// Pode conter lógica, cálculos, etc.

return valorRetornado;  
}

function saudacao(nome) {  
 return "Olá, " + nome + "!";  
}  
  
let mensagem = saudacao("Maria");  
console.log(mensagem); // Saída: Olá, Maria!

Outras características:

* **Escopo:**

Funções criam seus próprios escopos, onde as variáveis declaradas dentro delas são acessíveis apenas dentro da função.

* **Reutilização:**

Funções podem ser chamadas várias vezes em diferentes partes do código, evitando repetição.

* **Parâmetros e Argumentos:**

Parâmetros são variáveis declaradas na definição da função, enquanto argumentos são os valores passados para a função quando ela é chamada.

* **Retorno:**

A palavra-chave return especifica o valor que a função retorna quando é chamada.

### Função expressa

const somar = function(a, b) {  
 return a + b;  
};  
  
let resultado = somar(5, 3);  
console.log(resultado); // Saída: 8

### Função de seta

const multiplicar = (a, b) => a \* b;  
  
let resultadoMultiplicacao = multiplicar(4, 6);  
console.log(resultadoMultiplicacao); // Saída: 24

## DOM – Document Object Model

DOM, ou Modelo de Objeto de Documento, é uma interface de programação que representa um documento HTML, XML ou SVG como uma árvore de objetos. Em termos simples, o DOM permite que o JavaScript acesse e manipule a estrutura, o conteúdo e o estilo de uma página web.

### O que é o DOM?

* O DOM é uma interface de programação que representa um documento web (HTML, XML, SVG) como uma árvore de objetos.
* Cada elemento da página, como um parágrafo, uma imagem ou um título, é representado como um nó na árvore.
* O DOM permite que o JavaScript acesse e modifique esses nós, alterando a estrutura, o conteúdo e o estilo da página.
* O DOM não é exclusivo do JavaScript, mas é frequentemente usado em conjunto com ele para criar páginas web interativas.

### Relação entre DOM e JavaScript

* O JavaScript usa o DOM para acessar e manipular o conteúdo de uma página web.
* O DOM fornece a estrutura necessária para que o JavaScript possa interagir com a página.
* Por exemplo, o JavaScript pode usar o DOM para adicionar um novo parágrafo à página, alterar o texto de um título ou ocultar um elemento.

**Exemplos de uso do DOM com JavaScript:**

* document.getElementById("idDoElemento"): Seleciona um elemento HTML com base no seu ID.
* elemento.innerHTML = "Novo conteúdo": Altera o conteúdo HTML de um elemento.
* elemento.style.color = "red": Altera o estilo CSS de um elemento.
* document.createElement("p"): Cria um novo elemento HTML do tipo parágrafo.
* elemento.addEventListener("click", minhaFuncao): Adiciona um ouvinte de evento a um elemento.

Em resumo: O DOM é a representação estruturada de uma página web em forma de árvore de objetos, enquanto o JavaScript é a linguagem que permite manipular essa estrutura para tornar a página interativa.

<https://developer.mozilla.org/pt-BR/docs/Web/API/Document_Object_Model/Introduction>

## Metodos e Objetos

### parseFloat()

O método parseFloat() em JavaScript é utilizado para converter uma string em um número de ponto flutuante (decimal). Ele analisa a string fornecida e retorna o primeiro valor numérico encontrado, ignorando caracteres não numéricos que aparecem após o número. Se a string não puder ser convertida em um número, ele retorna NaN (Not a Number).

Como funciona:

**1. Análise da string:**

parseFloat() analisa a string da esquerda para a direita, buscando por um número válido.

**2. Interpretação:**

Ele interpreta a sequência de caracteres como um número decimal, incluindo o ponto decimal (.).

**3. Ignora caracteres não numéricos:**

Se encontrar um caractere que não faz parte de um número válido (ex: letras, espaços em branco), ele ignora esse caractere e os seguintes, retornando o valor numérico encontrado até aquele ponto.

**4. Tratamento de erros:**

Se a string não começar com um número válido, parseFloat() retorna NaN.

let str1 = "3.14";  
let num1 = parseFloat(str1); // num1 será 3.14

console.log(num1);  
  
let str2 = " 123.45abc ";  
let num2 = parseFloat(str2); // num2 será 123.45

console.log(num2);  
  
let str3 = "abc123.45";

let num3 = parseFloat(str3); // num3 será NaN

console.log(num3);  
  
let str4 = "Infinity";

let num4 = parseFloat(str4); // num4 será Infinity

console.log(num4);  
  
let str5 = "-Infinity";

let num5 = parseFloat(str5); // num5 será -Infinity

console.log(num5);

Usos comuns:

* **Conversão de dados de entrada:**

Quando você recebe dados de um formulário ou de uma fonte externa que são strings, mas precisam ser usados em cálculos numéricos, o parseFloat() pode ser útil para convertê-los em números.

* **Análise de valores de texto:**

Se você tem um texto que contém números e quer extrair esses números como valores numéricos, o parseFloat() pode ajudar.

Diferença entre parseFloat() e Number():

* parseFloat() é mais específico para números de ponto flutuante e ignora caracteres não numéricos após o início do número.
* Number() tenta converter a string para qualquer tipo numérico (inteiro ou float), e retorna NaN se não conseguir converter a string.

**Verificação de NaN:**

É importante verificar se o resultado de parseFloat() é NaN usando a função isNaN() para tratar casos onde a conversão falha, como mostrado no exemplo abaixo:

let str = "abc";  
let num = parseFloat(str);  
  
if (isNaN(num)) {  
 console.log("A string não pôde ser convertida em um número.");

} else {

console.log("O número convertido é:", num);  
}

### toFixed()

O método toFixed(n) em JavaScript formata um número como uma string, utilizando notação de ponto fixo e arredondando o número para n casas decimais. O resultado é sempre uma string, mesmo que o número original fosse um número.

numero.toFixed(n)

Onde:

* número é o objeto Number que você deseja formatar.
* n é um número inteiro entre 0 e 20 (inclusive), especificando o número de casas decimais desejadas. Se n for omitido, assume-se 0.

let num1 = 123.456;  
let num2 = 5.67;

console.log(num1.toFixed(2)); *// Saída: "123.46" (arredondamento para duas casas decimais)*

console.log(num2.toFixed(0)); // Saída: "6" (arredondamento para zero casas decimais)

console.log(num1.toFixed()); // Saída: "123" (sem casas decimais)

**Observações:**

* O método toFixed() arredonda o número para o valor mais próximo com a quantidade de casas decimais especificada.
* Se o número de casas decimais for maior do que o número de casas decimais do número original, zeros serão adicionados ao final da string resultante para preencher o espaço.
* O método toFixed() retorna uma string, não um número. Para realizar cálculos com o resultado, você precisará converter a string de volta para um número usando Number(), por exemplo: Number(num1.toFixed(2)).
* Se o número for muito grande ou muito pequeno, o método toFixed() pode retornar uma string em notação exponencial, dependendo da implementação do navegador.

**Usos comuns:**

* Formatando valores monetários para exibição ao usuário.
* Garantindo que números tenham um número fixo de casas decimais para fins de exibição ou comparação.
* Evitando notação científica em números grandes ou pequenos.

### Math

Em JavaScript, o objeto Math é um objeto global que contém constantes e métodos para operações matemáticas. Ele não é um objeto de função e não pode ser instanciado com o operador new. Todos os membros de Math são estáticos.

**Constantes Matemáticas:**

O objeto Math possui várias constantes matemáticas úteis, como:

* + Math.E: O número de Euler (aproximadamente 2.718).
  + Math.LN10: Logaritmo natural de 10.
  + Math.LN2: Logaritmo natural de 2.
  + Math.LOG10E: Logaritmo de E na base 10.
  + Math.LOG2E: Logaritmo de E na base 2.
  + Math.PI: O valor de Pi (aproximadamente 3.14159).
  + Math.SQRT1\_2: Raiz quadrada de 1/2.
  + Math.SQRT2: Raiz quadrada de 2.

**Funções Matemáticas:**

O objeto Math oferece uma variedade de funções matemáticas, incluindo:

* + **Arredondamento:**
    - Math.round(x): Arredonda para o inteiro mais próximo.
    - Math.floor(x): Arredonda para baixo para o inteiro mais próximo.
    - Math.ceil(x): Arredonda para cima para o inteiro mais próximo.
  + **Cálculo:**
    - Math.abs(x): Retorna o valor absoluto.
    - Math.pow(x, y): Retorna x elevado à potência de y.
    - Math.sqrt(x): Retorna a raiz quadrada.
    - Math.cbrt(x): Retorna a raiz cúbica.
    - Math.exp(x): Retorna e elevado à potência de x.
    - Math.log(x): Retorna o logaritmo natural.
    - Math.log10(x): Retorna o logaritmo na base 10.
    - Math.log2(x): Retorna o logaritmo na base 2.
  + **Trigonometria:**
    - Math.sin(x): Retorna o seno.
    - Math.cos(x): Retorna o cosseno.
    - Math.tan(x): Retorna a tangente.
    - Math.asin(x): Retorna o arco seno.
    - Math.acos(x): Retorna o arco cosseno.
    - Math.atan(x): Retorna o arco tangente.
    - Math.atan2(y, x): Retorna o ângulo em radianos.
  + **Outros:**
    - Math.random(): Retorna um número pseudoaleatório entre 0 (inclusivo) e 1 (exclusivo).
    - Math.max(x, y, z, ...): Retorna o maior valor.
    - Math.min(x, y, z, ...): Retorna o menor valor.
    - Math.hypot(x, y, z, ...): Retorna a raiz quadrada da soma dos quadrados.