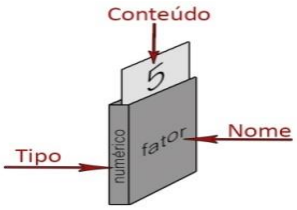
Variáveis

Em programação, variáveis são espaços de armazenamento nomeados que podem conter diferentes tipos de dados, como números, texto, valores booleanos (verdadeiro ou falso) e até mesmo estruturas de dados mais complexas. As variáveis permitem que os programadores guardem e manipulem informações de forma dinâmica durante a execução de um programa.

Imagine uma variável como uma caixa com um rótulo, onde você pode colocar algo dentro e depois pegá-lo quando precisar.

Essa "caixa" é uma área da memória do computador que tem um nome, e você pode armazenar valores nela, modificá-los e recuperá-los quando necessário.



As variáveis são usadas para:

- Armazenar Dados: Você pode guardar valores temporariamente para usar mais tarde. Por exemplo, um programa pode armazenar a idade de uma pessoa ou o nome de um usuário.
- Realizar Cálculos: As variáveis podem conter números que são usados em cálculos ou operações matemáticas.
- Alterar o Fluxo do Programa: Variáveis podem ser usadas em estruturas de decisão e loops para controlar o comportamento do programa com base nas condições.
- Comunicação: Variáveis permitem que partes diferentes de um programa compartilhem informações entre si.

Aqui está um exemplo simples de criação e uso de uma variável em JavaScript:

```
// Declarar uma variável chamada "idade" e atribuir o valor 25 a ela
let idade = 25;
// Imprimir o valor da variável no console
console.log(idade); // Isso exibirá "25" no console
```

Neste exemplo, a variável chamada "idade" armazena o valor 25, que pode ser acessado e usado posteriormente no programa. Variáveis são fundamentais para a programação,

pois permitem que os programas sejam flexíveis e capazes de lidar com diferentes dados e situações.

Declaração de variáveis

A declaração de variáveis é o processo de criar uma variável e associar um nome a ela em um programa de computador. É como dizer ao computador: "Aqui está uma área de memória que vamos chamar de 'nome' e onde vamos armazenar algum valor."

Em JavaScript, você pode criar uma variável usando uma das três palavras-chave de declaração de variáveis: **var**, **let** ou **const**. A escolha entre essas palavras-chave depende do escopo e da mutabilidade que você deseja para a variável.

Aqui estão exemplos de como criar variáveis usando cada uma dessas palavras-chave:

Usando **var** (antigo, evite usar em cenários modernos):

```
// Declara uma variável chamada "idade" e atribui o valor 25 a ela
var idade = 25;
```

Usando **let** (recomendado para variáveis que podem mudar de valor):

```
// Declara uma variável chamada "nome" e atribui o valor "João" a ela
let nome = "João";
```

Usando **const** (para variáveis com valores constantes, que não serão reatribuídos):

```
// Declara uma variável chamada "PI" e atribui o valor 3.14159 a ela
const PI = 3.14159;
```

Lembre-se de que as variáveis em JavaScript são sensíveis a maiúsculas e minúsculas, ou seja, "idade", "Idade" e "IDADE" seriam consideradas nomes de variáveis diferentes.

Após a declaração de variáveis, você pode atribuir valores a elas, alterar esses valores, usá-las em cálculos e realizar outras operações com elas dentro do seu programa. As variáveis são uma parte essencial da programação, pois permitem que você armazene e manipule dados de maneira flexível.

Exemplos de utilização de variáveis

Let

```
let sobrenome = 'Dionisio';
```

```
console.log(sobrenome);
```

Observação: Devemos evitar a utilização da palavra reservada **“var”** para declarar variáveis.

Constantes

São como variáveis em que uma vez definido o valor, este não pode ser mudado pelo programa. Exemplo:

```
const PI = 3.1415;
```

Por convenção e para diferenciar uma variável normal de uma constante, ela deve ser escrita com letra maiúscula.

Outro ponto é que uma constante deve sempre ser inicializada na declaração, caso contrário, será gerado um erro.

```
const PI = 3.14;
console.log(PI);
```

Comentário

Comentários são trechos de texto que os programadores incluem em seu código para fornecer explicações, notas ou informações sobre o que o código está fazendo.

Comentários não são executados pelo computador e são usados apenas para fins de documentação e compreensão do código.

Em JavaScript, você pode criar comentários de duas maneiras principais: comentários de linha única e comentários de várias linhas.

Comentários de Linha Única:

```
// Isto é um comentário de linha única.
// O computador não executa essa parte do texto.
let idade = 25; // Aqui estamos declarando uma variável chamada "idade" com o valor 25.
```

Comentários de Múltiplas Linhas:

```
/*
Este é um comentário de múltiplas linhas.
Pode ser usado para explicar partes maiores do código
ou fazer anotações mais detalhadas.
*/
let nome = "Anna";
```

Comentários são úteis para explicar o propósito de uma seção de código, desativar temporariamente partes do código (comentários de depuração), ou fornecer dicas e notas para outros programadores (ou até mesmo para você mesmo no futuro) que possam revisar ou trabalhar com o código.

Lembre-se de que, embora comentários sejam ignorados pelo computador durante a execução, eles desempenham um papel importante na clareza e manutenção do código, tornando-o mais compreensível e fácil de entender.

Operadores aritméticos

Os operadores aritméticos em JavaScript são símbolos especiais usados para realizar operações matemáticas em valores numéricos (números).

Eles permitem que você realize cálculos, como adição, subtração, multiplicação, divisão e muito mais. Aqui estão os principais operadores aritméticos em JavaScript:

Adição (+):
Realiza a adição de dois valores.

```
let soma = 5 + 3; // Resultado: 8
```

Subtração (-):
Realiza a subtração de dois valores.

```
let  diferenca = 10 - 6; // Resultado: 4
```

Multiplicação (*):
Realiza a multiplicação de dois valores.

```
let produto = 4 * 3; // Resultado: 12
```

Divisão (/):
Realiza a divisão de dois valores.

```
let quociente = 20 / 5; // Resultado: 4
```

Módulo (%):
Retorna o resto da divisão entre dois valores.

```
let resto = 10 % 3; // Resultado: 1 (resto da divisão de 10 por 3)
```

Incremento (++):
Incrementa o valor de uma variável por 1.

```
let numero = 5;
numero++; // Agora numero é 6
```

Decremento (--):
Decrementa o valor de uma variável por 1.

```
let numero = 8;
numero--; // Agora numero é 7
```

Os operadores aritméticos podem ser usados em expressões mais complexas e combinados com outros operadores para realizar cálculos mais elaborados. Lembre-se da ordem de precedência dos operadores (Parecido com a matemática básica), que pode ser alterada usando parênteses para controlar a ordem em que as operações são executadas.

Exemplo de uso de operadores aritméticos em uma expressão:

```
let resultado = (10 + 5) * 2 - 3 / 2; // Resultado: 21.5
```

Os operadores aritméticos são essenciais para realizar cálculos em programas e para manipular dados numéricos de maneira eficaz.

Operador de atribuição

Um **operador de atribuição** em JavaScript é usado para atribuir um valor a uma variável. Ele permite que você coloque um valor dentro de uma variável para que possa ser usado posteriormente no programa.

O operador de atribuição mais comum é o sinal de igual (=).

Aqui está um exemplo simples de como um operador de atribuição é usado em JavaScript:

```
// Atribui o valor "Maria" à variável chamada "nome"
let nome = "Maria";
```

Neste exemplo, o operador de atribuição (=) é usado para colocar o valor "Maria" dentro da variável chamada "nome".

Isso significa que você pode usar a variável "nome" posteriormente em seu código para acessar o valor "Maria".

Além do operador de atribuição simples (=), existem também operadores de atribuição combinados que realizam uma operação e, em seguida, atribuem o resultado a uma variável. Por exemplo:

```
let numero = 10;
numero += 5; // Isso é equivalente a: numero = numero + 5; Resultado: 15
```

Nesse caso, o operador de atribuição combinado (+=) adiciona 5 ao valor existente da variável "numero" e, em seguida, atribui o novo valor (15) de volta à variável "numero".

Os operadores de atribuição são fundamentais para manipular dados em um programa, pois permitem que você armazene e atualize valores em variáveis, tornando o código mais dinâmico e interativo.

Exemplos de operadores de atribuição combinados

Operador de Atribuição e Adição (+=):

```
let numero = 10;
numero += 5; // Agora "numero" é igual a 15
```

Operador de Atribuição e Subtração (-=):

```
let total = 50;
total -= 20; // Agora "total" é igual a 30
```

Operador de Atribuição e Multiplicação (*=):

```
let quantidade = 4;
quantidade *= 3; // Agora "quantidade" é igual a 12
```

Operador de Atribuição e Divisão (/=):

```
let valor = 100;
valor /= 2; // Agora "valor" é igual a 50
```

Operador de Atribuição e Módulo (%=):

```
let numero = 15;
numero %= 4; // Agora "numero" é igual a 3 (resto da divisão de 15 por 4)
```

Operador de Atribuição e Exponenciação (=) (disponível em versões mais recentes do JavaScript):**

```
let base = 2;
base **= 3; // Agora "base" é igual a 8 (2 elevado à potência de 3)
```

Esses operadores de atribuição combinados realizam uma operação aritmética com o valor atual da variável e o valor à direita do operador, e então atribuem o resultado de volta à variável.

Isso pode tornar seu código mais conciso e legível, especialmente quando você está realizando operações comuns de atribuição e cálculo simultaneamente.

Casting

O que é casting em programação?

Quando falamos de CASTING em programação estamos nos referindo à conversão de um tipo de dado para outro tipo. Isso é necessário quando você deseja realizar operações ou manipulações entre diferentes tipos de dados que não são diretamente compatíveis.

Imagine que você tem duas variáveis de tipos diferentes, por exemplo, números e texto. Em programação, às vezes você precisa converter, ou "fazer um casting", de um tipo de dado em outro para que possa trabalhar com eles juntos.

Existem duas formas principais de casting:

Casting Implícito (Conversão Implícita):

Nesse caso, a linguagem de programação realiza automaticamente a conversão de um tipo de dado para outro em certas situações. Por exemplo, quando você soma um número inteiro com um número de ponto flutuante, a linguagem pode realizar um casting implícito para que ambos os números tenham o mesmo tipo antes da operação.

Exemplo de casting Implícito:

```
let numeroInteiro = 5;
let numeroDecimal = 2.5;

// O número inteiro é automaticamente convertido para decimal antes da soma.
let resultado = numeroInteiro + numeroDecimal;

// O resultado será 7.5
console.log(resultado);
```

Casting Explícito (Conversão Explícita):

O **casting explícito** envolve a conversão deliberada de um tipo de dado para outro usando funções ou operadores específicos. Geralmente, você faz isso quando precisa garantir que os tipos de dados sejam compatíveis antes de realizar uma operação.

Exemplo de casting Explícito:

```
let numeroTexto = "10";

// Convertendo o texto para um número inteiro usando parseInt.
let numero = parseInt(numeroTexto);

// O resultado será 10
console.log(numero);
```

Lembre-se de que nem todas as linguagens de programação permitem todas as formas de casting, e em alguns casos, casting incorreto ou inadequado pode levar a erros ou resultados inesperados. Portanto, é importante entender como as conversões funcionam na linguagem que você está usando e usá-las com cuidado.

Exemplos de casting explicito

Casting de Número para Texto:

Suponha que você tem um número e quer exibi-lo junto com um texto. Você precisa transformar o número em texto para fazer isso:

```
let numero = 42;

// Aqui, o número é convertido em texto automaticamente.
let texto = "O número é: " + numero;
console.log(texto);
```

Também podemos fazer o casting explícito de um número para um string, observe o exemplo.

```
let numero = 45

// casting explícito
let texto = numero.toString()
console.log(texto, typeof texto);
```

Casting de texto para número:

Às vezes, você pode precisar fazer cálculos com texto que representa números. Nesse caso, você precisa transformar o texto em número:

```
let texto = "10";

// Aqui, o texto é convertido em um número inteiro
let numero = parseInt(texto);

// Agora você pode fazer cálculos com o número.
let soma = numero + 5;
console.log(soma);
```

Casting de Ponto Flutuante (Decimal):

Similar ao exemplo anterior, mas para números com casas decimais:

```
let texto = "3.14";
```

```
// Aqui, o texto é convertido em um número de ponto flutuante.
let numero = parseFloat(texto);

let resultado = numero * 2;
console.log(resultado);
```

Lembre-se de que nem todos os tipos de casting são tão diretos e fáceis. Alguns tipos de conversão podem envolver considerações especiais para lidar com diferentes formatos e limitações. Mas a ideia geral é sempre a mesma: você está ajustando os tipos de dados para que possam funcionar juntos da maneira que você precisa.

TypeOf

O que é e o que faz o comando typeof?

Em JavaScript, o comando **typeof** é um operador unário que é usado para determinar o tipo de dado de uma expressão ou variável.

Ele retorna uma string que representa o tipo do valor do conteúdo avaliado.

O operador **typeof** é frequentemente usado para verificar o tipo de dado antes de executar operações específicas ou para fazer tratamento condicional de acordo com o tipo.

A sintaxe básica do typeof é a seguinte:

```
typeof valor
```

Alguns exemplos de uso do typeof:

```
let numero = 5;
let texto = "Olá";
let array = [1, 2, 3];
let objeto = { nome: "Alice", idade: 30 };

console.log(typeof numero);     // Saída: "number"
console.log(typeof texto);      // Saída: "string"
console.log(typeof array);      // Saída: "object"
console.log(typeof objeto);     // Saída: "object"
```

É importante observar que o **typeof** nem sempre retorna valores exatamente intuitivos. Por exemplo, **typeof null** retorna "**object**", o que é uma característica histórica e pode ser considerado um erro de design da linguagem.

Para verificar se uma variável é de um tipo específico, é uma boa prática usar o **typeof** junto com instruções condicionais, como um if, por exemplo:

```
let valor = "Olá";

if (typeof valor === "string") {
  console.log("A variável é uma string.");
} else {
  console.log("A variável não é uma string.");
}
```

O typeof é uma ferramenta útil para lidar com diferentes tipos de dados em JavaScript e para evitar erros ao realizar operações que são específicas de um determinado tipo de dado.

Funções matemáticas básicas

Math.pow

Em JavaScript, o Math.pow() é uma função embutida que permite calcular a potência de um número, permitindo elevar um número a uma certa potência.

A sintaxe básica do Math.pow() é a seguinte:

```
Math.pow(base, expoente)
```

Onde:

- base: O número que você deseja elevar à potência.
- expoente: O valor da potência ao qual a base será elevada.

O resultado retornado por Math.pow() é a base elevada ao expoente. Veja alguns exemplos:

```
// 2 elevado à potência 3 = 8
let resultado1 = Math.pow(2, 3);
console.log(resultado1); // Saída: 8

// 5 elevado à potência 2 = 25
let resultado2 = Math.pow(5, 2);
console.log(resultado2); // Saída: 25
```

Além do Math.pow(), você também pode usar o operador ** para realizar cálculos de potência em JavaScript a partir do ECMAScript 2016:

```
// 3 elevado à potência 4 = 81
let resultado3 = 3 ** 4;
console.log(resultado3); // Saída: 81
```

Usar Math.pow() ou o operador ** é uma maneira conveniente de calcular potências em JavaScript, e eles são úteis em várias situações, como cálculos matemáticos, processamento de dados científicos e muito mais.

Math.sqrt

Em JavaScript, o Math.sqrt() é uma função incorporada que é usada para calcular a raiz quadrada de um número. Essa função é parte do objeto Math, que fornece várias funções matemáticas para realizar cálculos.

A sintaxe básica da função Math.sqrt() é a seguinte:

```
Math.sqrt(numero)
```

Aqui, numero é o valor numérico do qual você deseja calcular a raiz quadrada. O resultado retornado pela função é a raiz quadrada desse número.

Vejamos alguns exemplos de uso do Math.sqrt():

```
let numero1 = 9;
let resultado1 = Math.sqrt(numero1);
// Saída: 3, pois a raiz quadrada de 9 é 3.
console.log(resultado1);

let numero2 = 25;
let resultado2 = Math.sqrt(numero2);
// Saída: 5, pois a raiz quadrada de 25 é 5.
console.log(resultado2);

// Saída: ~1.414, pois a raiz quadrada de 2 é aproximadamente 1.414.
let numero3 = 2;
let resultado3 = Math.sqrt(numero3);
console.log(resultado3);
```

O Math.sqrt() é muito útil em cálculos que envolvem geometria, física, estatísticas e muitos outros campos matemáticos. Lembre-se de que o resultado pode ser um número de ponto flutuante, mesmo que o número original seja um inteiro.

Em resumo, o Math.sqrt() em JavaScript é uma função que ajuda a calcular a raiz quadrada de um número, permitindo que você realize operações matemáticas mais complexas em seus programas.

Math.cbrt

O comando Math.cbrt em JavaScript é uma função da biblioteca Math que é usada para calcular a raiz cúbica de um número. A raiz cúbica de um número x é aquele número y que, quando elevado ao cubo, resulta em x.

A sintaxe básica do **Math.cbrt** é a seguinte:

```
Math.cbrt(x)
```

Aqui, x é o número do qual você deseja calcular a raiz cúbica. A função retorna a raiz cúbica desse número.

Exemplos de uso do **Math.cbrt**:

```
let numero = 27;
let raizCubica = Math.cbrt(numero);
console.log(raizCubica); // Saída: 3

let numeroNegativo = -8;
let raizCubicaNegativa = Math.cbrt(numeroNegativo);
console.log(raizCubicaNegativa); // Saída: -2
```

A função **Math.cbrt** é útil quando você precisa calcular raízes cúbicas com precisão.

Math.PI

O comando **Math.PI** em JavaScript é uma propriedade do objeto Math que representa o valor da constante matemática **Pi (π).** O valor de Pi é a relação entre a circunferência de qualquer círculo e seu diâmetro. Essa constante é amplamente usada em cálculos matemáticos e científicos que envolvem geometria, trigonometria e outras áreas.

A propriedade **Math.PI** é um número de ponto flutuante aproximado de **3.141592653589793**, mas você pode usá-lo com essa precisão para a maioria das aplicações cotidianas.

Aqui estão alguns exemplos de como você pode usar **Math.PI** em JavaScript:

```
// Calculando a circunferência (perímetro) de um círculo com um raio de 5 unidades
let raio = 5;
let circunferencia = 2 * Math.PI * raio;
console.log("A circunferência é: " + circunferencia);

// Calculando a área de um círculo com um raio de 3 unidades
let raioCirculo = 3;
let areaCirculo = Math.PI * raioCirculo * raioCirculo;
console.log("A área do círculo é: " + areaCirculo);
```

O uso de **Math.PI** facilita muito os cálculos relacionados a círculos e outras formas circulares, permitindo que você realize operações matemáticas de maneira mais precisa e eficiente.

Math.abs

O comando **Math.abs()** em JavaScript é uma função que retorna o valor absoluto de um número, ou seja, o valor positivo desse número, removendo qualquer sinal negativo.

A sintaxe básica da função **Math.abs**() é a seguinte:

```
Math.abs(numero)
```

Aqui, numero é o valor numérico para o qual você deseja obter o valor absoluto.

Alguns exemplos de uso do **Math.abs():**

```
console.log(Math.abs(-5));   // Saída: 5
console.log(Math.abs(10));   // Saída: 10
console.log(Math.abs(-3.14)); // Saída: 3.14
```

O **Math.abs**() é útil quando você quer garantir que um número seja positivo, independentemente do sinal original. Isso é especialmente útil ao lidar com cálculos onde o sinal do número não é importante, como encontrar a distância entre dois pontos em um gráfico, por exemplo:

```
let pontoA = -7;
let pontoB = 4;

let distancia = Math.abs(pontoB - pontoA);
console.log("A distância entre os pontos é: " + distancia);
```

Nesse exemplo, distância será calculada como 11, independentemente dos sinais originais dos pontos A e B. Isso ocorre porque o **Math.abs**() garante que o cálculo da distância seja sempre positivo.

Math.ceil

Em JavaScript, o comando **Math.ceil()** é uma função da biblioteca interna Math que é usada para arredondar um número para **cima**, ou seja, para o próximo inteiro maior ou igual ao número fornecido como argumento.

Aqui está a sintaxe básica da função **Math.ceil**():

```
Math.ceil(numero);
```

Onde numero é o valor que você deseja arredondar para cima.

Exemplos de uso do **Math.ceil():**

```
console.log(Math.ceil(4.2));  // Saída: 5
console.log(Math.ceil(7.7));  // Saída: 8
console.log(Math.ceil(9.1));  // Saída: 10
console.log(Math.ceil(-3.6)); // Saída: -3 (pois é o próximo inteiro maior ou igual a -3.6)
```

Note que mesmo números negativos podem ser arredondados para cima para se tornarem números inteiros maiores do que o número original.

O Math.ceil() é útil quando você precisa garantir que um valor seja arredondado para cima, por exemplo, ao lidar com medidas, quantidades ou valores financeiros onde a fração precisa ser arredondada para a unidade superior.

Math.floor

O comando Math.floor em JavaScript é uma função que arredonda um número para o próximo número inteiro menor ou igual ao número original. Isso significa que, ao usar Math.floor, você sempre obtém o maior número inteiro que seja menor ou igual ao número que está sendo arredondado.

Aqui está a sintaxe básica da função Math.floor:

```
Math.floor(numero);
```

Onde numero é o valor que você deseja arredondar.

Vamos ver alguns exemplos para entender melhor como o Math.floor funciona:

```
console.log(Math.floor(3.7));  // Saída: 3
console.log(Math.floor(7.2));  // Saída: 7
console.log(Math.floor(9.9));  // Saída: 9
```

Essa função é especialmente útil quando você precisa garantir que um número seja arredondado para baixo, para obter um valor inteiro. Por exemplo, ao trabalhar com *índices de arrays* ou em situações em que é importante obter um valor que esteja "*alinhado*" com uma unidade específica.

Lembre-se de que Math.floor sempre arredonda para baixo, o que pode levar a resultados inesperados se você estiver procurando arredondar para o número inteiro mais próximo. Nesse caso, você pode usar Math.round para arredondamento normal ou Math.ceil para arredondamento para cima.

Math.round

Em JavaScript, o comando Math.round() é uma função que arredonda um número para o número inteiro mais próximo. Ele segue a regra matemática padrão de arredondamento, onde valores decimais iguais ou maiores que 0.5 são arredondados para cima, enquanto valores menores que 0.5 são arredondados para baixo.

Aqui está a sintaxe básica do Math.round():

```
Math.round(numero)
```

Onde numero é o valor decimal que você deseja arredondar para o número inteiro mais próximo.

Exemplos de uso do Math.round():

```
let valor1 = 5.3;
let valor2 = 7.8;

let arredondado1 = Math.round(valor1);
let arredondado2 = Math.round(valor2);

console.log(arredondado1); // Saída: 5
console.log(arredondado2); // Saída: 8
```

Lembre-se de que o Math.round() sempre retorna um número inteiro, independentemente de o valor original ser positivo ou negativo.

```
let valorNegativo1 = -3.7;
let valorNegativo2 = -1.2;

let arredondadoNegativo1 = Math.round(valorNegativo1);
let arredondadoNegativo2 = Math.round(valorNegativo2);

console.log(arredondadoNegativo1); // Saída: -4
console.log(arredondadoNegativo2); // Saída: -1
```

O Math.round() é útil quando você precisa trabalhar com números inteiros em situações em que valores decimais precisam ser aproximados para a unidade mais próxima.

Math.random

Em JavaScript, o comando Math.random() é uma função que gera um número decimal pseudoaleatório no intervalo entre 0 (incluído) e 1 (não incluído).

Aqui está a sintaxe básica da utilização do Math.random():

```
let numeroAleatorio = Math.random();
```

Para gerar números aleatórios em um intervalo diferente de 0 a 1, você pode usar essa fórmula:

```
let numeroAleatorioNoIntervalo = Math.random() * (max - min) + min;
```

Onde **max** é o valor máximo desejado (não incluído) e **min** é o valor mínimo desejado.

Exemplo de geração de número aleatório entre 1 e 10:

```
let numeroAleatorioEntre1e10 = Math.random() * (10 - 1) + 1;
```

Este é um método básico para obter números aleatórios em JavaScript. No entanto, é importante entender que os números gerados por Math.random() não são realmente "*verdadeiramente*" aleatórios, mas sim pseudorandômicos.

Isso significa que eles são gerados por algoritmos matemáticos e podem se repetir após um certo período. Se você precisa de números mais aleatórios e seguros para fins criptográficos, por exemplo, é recomendado buscar bibliotecas externas especializadas em geração de números aleatórios seguros.

Para fazer com que o resultado de Math.random() seja um número inteiro, você pode combinar a função Math.floor() ou outras técnicas de arredondamento:

```
// Gera um número inteiro entre 0 e 9.
let numeroInteiroAleatorio = Math.floor(Math.random() * 10);
```

Lembre-se de que o uso de Math.random() é uma maneira simples e rápida de gerar números aleatórios para muitos cenários, mas pode não ser adequado para aplicações que requerem alta qualidade e segurança na aleatoriedade.

Math.max e Math.min

Math.max:

O método **Math.max** é usado para encontrar o maior valor entre um ou mais números passados como argumentos. A sintaxe é a seguinte:

```
Math.max(numero1, numero2, ...);
```

Aqui, você pode passar vários números separados por vírgulas e o método retornará o maior valor entre eles.

Exemplo:

```
let maiorNumero = Math.max(10, 5, 20, 8);
console.log(maiorNumero); // Saída: 20
```

Math.min:

O método Math.min é semelhante ao Math.max, mas em vez de retornar o maior valor, ele retorna o menor valor entre um conjunto de números. A sintaxe é a seguinte:

```
Math.min(numero1, numero2, ...);
```

Exemplo:

```
let menorNumero = Math.min(10, 5, 20, 8);
console.log(menorNumero); // Saída: 5
```

É importante notar que esses métodos aceitam qualquer número de argumentos, então você pode passar quantos números desejar. Além disso, eles também podem ser usados com variáveis ou expressões numéricas, não apenas com valores literais.

Exemplo com variáveis:

```
let a = 15;
let b = 25;
let c = 5;

let maiorValor = Math.max(a, b, c);
console.log(maiorValor); // Saída: 25
```

Esses métodos são úteis quando você precisa determinar rapidamente o maior ou o menor valor em um conjunto de números, como ao lidar com dados em um array ou ao calcular limites em algoritmos.

Exercícios

Sempre que possível utilize variáveis para a resolução do problema e utilize o comando 'console.log' para exibir o resultado no console.

Exercício 11:

Elaborar um algoritmo que imprima em modo console a frase abaixo:

"Aprendendo Algoritmo"

Exercício 12:

Elabore um algoritmo que imprima a frase da maneira descrita abaixo, uma frase abaixo da outra (imprima no modo console):

Aprendendo Algoritmo
e Fazendo muito Exercício
Primeiro fazendo exercício em 'JavaScript'

Exercício 13:

Crie um arquivo HTML e vincule um arquivo JavaScript ao arquivo.
No arquivo JavaScript crie duas variáveis, uma para armazenar o seu nome e outra para armazenar sua idade.
Exiba estes dados no console da página Web.

Exercício 14:

Crie uma aplicação que receba duas variáveis do tipo inteiro, exiba os valores digitados e posteriormente exiba a primeira variável acrescida de uma unidade e a segunda variável decrescida de uma unidade.

Exercício 15:

Crie uma aplicação que receba 5 números e exiba a soma com a seguinte frase:

"Os números digitados foram ..., ..., ..., ... e sua soma é ... .

Exercício 16:

Cria uma aplicação que receba dois números e exibir as seguintes mensagens:

O números digitados foram ... e .... .
A soma dos números ... e ... é ... .
A subtração dos números ... e ... é ... .
A multiplicação dos números ... e ... é ... .
A divisão dos números ... e ... é ... .
A média dos números ... e ... é ... .

Exercício 17:

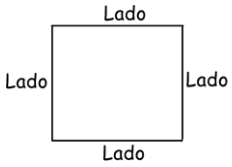
Crie uma aplicação que receba um número inteiro e imprimir seu antecessor e seu sucessor.

SAÍDA:
O número digitado foi ..., seu antecessor é ... e seu sucessor é ...

Exercício 18:

Crie uma aplicação para calcular a área e o perímetro de um quadrado.

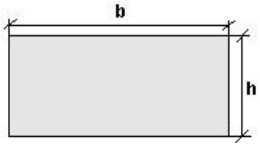
Área = lado * lado
Perímetro = é a soma de todos os lados



Exercício 19:

Elabore uma aplicação para calcular a área e o perímetro de um retângulo.

Área = b x h
Perímetro = é a soma de todos os lados



Exercício 20: Caça ao Tesouro

Imagine que você encontrou um mapa do tesouro com instruções para chegar ao local. A primeira instrução diz para andar 10 passos para o norte e depois 5 passos para o leste. Quantos passos você terá que dar no total?

Exercício 21: Festa de Aniversário

Você está organizando uma festa de aniversário e convidou 20 amigos. Cada amigo vai trazer dois presentes. Quantos presentes você vai receber no total?

Exercício 22: Venda de Sorvetes

Em um dia quente, você vendeu 50 sorvetes de baunilha e 30 sorvetes de chocolate. Cada sorvete custa R$3,50. Quanto dinheiro você ganhou?

Exercício 23: Idade do Cachorro

Você tem um cachorro que tem 3 anos de idade. Sabendo que a cada ano de vida do cachorro equivale a 7 anos de vida de um humano, quantos anos o cachorro tem em anos humanos?

Exercício 24: Coleta de Frutas

Você está colhendo maçãs em uma fazenda. Você já colheu 15 maçãs e pretende colher mais 10. Quantas maçãs você terá ao final?

Exercício 25: Economizando Mesada

Você recebeu uma mesada de R$50,00 e decidiu economizar R$15,00. Quanto dinheiro você terá sobrando?

Exercício 26: Contagem de Estrelas

Você está observando o céu à noite e já viu 8 estrelas. Depois, viu mais 5 estrelas. Quantas estrelas você viu no total?

Exercício 27: Economizando para o Lanche

Você quer economizar para comprar um lanche que custa R$7,50. Até agora, você já economizou R$3,20. Quanto mais você precisa economizar?

Exercício 28: Compartilhando Balas

Você tem 24 balas e quer dividir igualmente entre 6 amigos. Quantas balas cada amigo receberá?

Exercício 29: Plantando Flores

Você vai plantar flores em um jardim. Já plantou 9 flores e planeja plantar mais 12. Quantas flores terá no total?

Exercício 30: Conta de Restaurante

Peça ao usuário para inserir o valor total da conta do restaurante e o percentual de gorjeta. Calcule e exiba o valor total a ser pago.

Exercício 31: Comprando Livros

Peça ao usuário para inserir o preço de um livro. Calcule o valor total da compra com desconto de 10%.

Exercício 32: Qual é o erro

Analise o código a seguir e diga qual se o ele apresenta um erro. Se ele apresentar um erro diga qual é o erro.
Observação: Você não deve utilizar o computador para encontrar o erro.

```
let numero1 = 10;
let numero2 = 5;
let resultado = numero1 + numero2;
console.log("O resultado da soma é: " resultado);
```

Respostas:

Resposta exercício 001 até 010:

https://github.com/dionisioR/algoritmo_javascript_exercicio_001_010

Resposta exercícios 32:

Erro de Sintaxe