

ALGORITMO

Não é a linguagem de programação que define o programador, mas sim sua lógica.

David Ribeiro Guilherme
(https://www.pensador.com/frases_de_programador/)

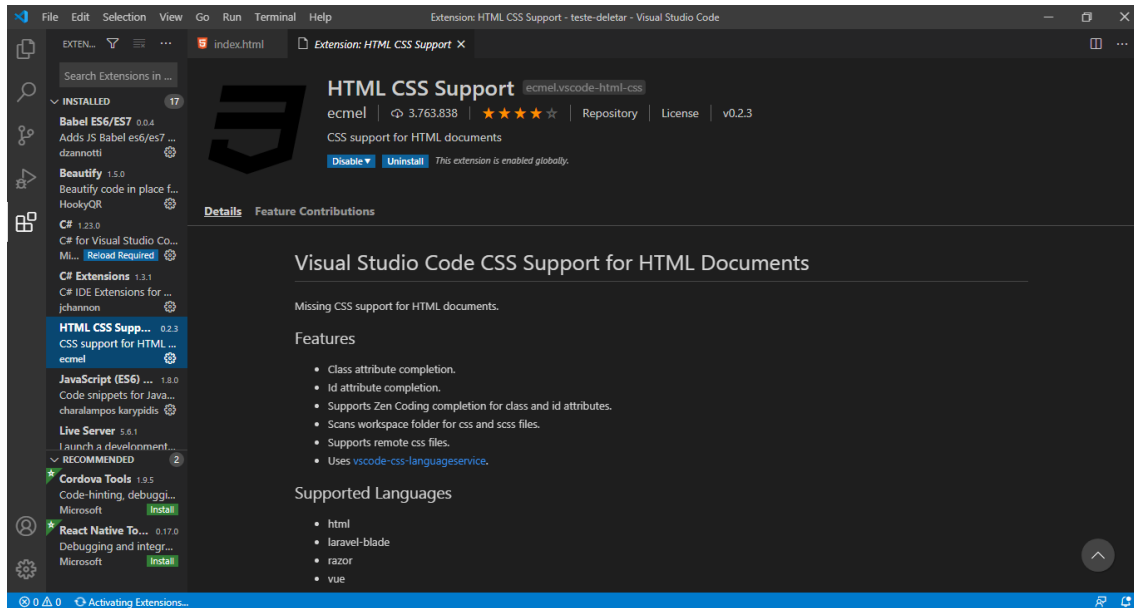
*Lógica de
Programação*

Sumário

Instalando o VS Code	5
Instalando as extensões.....	5
Node.JS	5
O que é Lógica?	6
O que é lógica de programação?	6
O que é linguagem de programação?.....	6
O que é Algoritmo?.....	7
Começando com a prática	8
Estrutura básica	8
Comandos de saída.....	8
Saída de dados no JavaScript	10
Console.Log	10
Explicando o console.log	10
Exemplos de como utilizar o console.log.....	10
Concatenação, Interpolação e Template String.....	11
Exercícios	11
Variáveis	12
Declaração de variáveis	13
Exemplos de utilização de variáveis.....	14
Let.....	14
Constantes.....	14
Comentário.....	14
Operadores aritméticos.....	15
Operador de atribuição	16
Exemplos de operadores de atribuição combinados.....	17
Exercícios	18
Casting	21
O que é casting em programação?	21
Exemplos de casting explícito	22
TypeOf	24
O que é e o que faz o comando typeof?	24
Funções matemáticas básicas	25
Math.pow	25

Math.sqrt.....	25
Math.cbrt.....	26
Math.PI	27
Math.abs.....	27
Math.ceil.....	28
Math.floor	29
Math.round	29
Math.random	30
Math.max e Math.min	31
Exercícios:	32
Operador de atribuição " = "	34
Incremento e decremento.....	35
Pré-incremento (++variável):	36
Pós-incremento (variável++):	36
Pré-decremento (--variável):	36
Pós-decremento (variável--):	37
Mas por que utilizar o ++ e não o += ou 'n = n+1'	37
Incorporando um arquivo JavaScript em uma página HTML	38
HTML	38
Exibindo informações no browser	38
Document Object Model (DOM).....	38
document.getElementById	40
innerHTML	41
Entrada de dados.....	42
Prompt.....	42
Exercícios:	43
Estruturas condicionais.....	45
Estrutura condicional IF..., IF... ELSE e IF... ELSE IF... ELSE	46
Estrutura Condicional "IF"	46
Estrutura Condicional "IF/ELSE"	47
IF... ELSE IF... ELSE	47
Respostas:.....	86
Links:.....	88

Instalando o VS Code



Instalando as extensões

- vscode-html-css
- auto-rename-tag
- LiveServer
- vscode-icons
- HTML CSS Support
- HTML end Tag Labels
- JavaScript (ES6) code snippets

Node.JS



O que é Lógica?

A lógica é a maneira como pensamos e organizamos nossas ideias para tomar decisões ou resolver problemas. É como seguir um caminho de raciocínio que faz sentido.

Imagine que você está jogando um jogo de adivinhação. Você recebe dicas e pistas, e precisa pensar sobre o que essas pistas significam para descobrir a resposta correta.

Isso é usar a lógica. Você está conectando as informações de forma inteligente para chegar a uma conclusão.

Na vida cotidiana, usamos a lógica o tempo todo. Quando fazemos planos, resolvemos quebra-cabeças, escolhemos entre opções ou entendemos como as coisas funcionam, estamos aplicando a lógica para tomar decisões e resolver desafios.

O que é lógica de programação?

A lógica de programação é uma maneira de pensar e estruturar as etapas ou instruções necessárias para resolver um problema por meio de um programa de computador.

É como criar um plano detalhado para que o computador possa entender e executar as ações corretas.

Pense nisso como dar instruções muito precisas para alguém que está seguindo um conjunto de regras bem definidas.

A lógica de programação envolve quebrar um problema complexo em partes menores, definir a ordem em que essas partes devem ser executadas e usar conceitos como repetição (fazer algo várias vezes) e decisão (escolher entre diferentes opções) para alcançar o resultado desejado.

Os programadores usam a lógica de programação para criar algoritmos, que são sequências específicas de passos que um computador pode entender e executar. É como escrever um roteiro para que o computador saiba exatamente o que fazer para resolver um problema ou realizar uma tarefa.

O que é linguagem de programação?

Linguagem de programação é um conjunto de regras e símbolos que os programadores usam para escrever instruções que um computador pode entender e executar.

Ela atua como uma ponte de comunicação entre humanos e máquinas, permitindo que você escreva um código para que o computador possa realizar uma ou várias tarefas específicas.

Existem muitas linguagens de programação diferentes, cada uma com suas próprias características e finalidades. Algumas linguagens são mais adequadas para tarefas

específicas, enquanto outras são mais versáteis e podem ser usadas em uma variedade de contextos. Aqui estão alguns exemplos populares de linguagens de programação:

Python: Conhecida por sua sintaxe legível e fácil de entender, é usada para desenvolvimento web, análise de dados, automação etc.

Java: Uma linguagem versátil usada em desenvolvimento de aplicativos, jogos, sistemas embarcados etc.

C++: Uma extensão da linguagem C que adiciona recursos de programação orientada a objetos, usada em desenvolvimento de software, jogos, sistemas de tempo real, entre outros.

JavaScript: Linguagem de script utilizada principalmente no desenvolvimento web para criar interações dinâmicas em páginas.

Ruby: Conhecida por sua simplicidade e produtividade, é usada para desenvolvimento web e automação.

C#: Desenvolvida pela Microsoft, é amplamente usada para criar aplicativos Windows e jogos usando a plataforma Unity.

PHP: Linguagem frequentemente usada para desenvolvimento web e criação de sites dinâmicos.

SQL: Linguagem de consulta de banco de dados usada para gerenciar e manipular dados em bancos de dados relacionais.

Esses são apenas alguns exemplos, e há muitas outras linguagens de programação por aí, cada uma com suas próprias vantagens e casos de uso específicos. Cada linguagem tem sua própria sintaxe e conjunto de regras, mas todas servem ao propósito de permitir que os programadores expressem instruções para o computador de maneira compreensível e eficaz.

O que é Algoritmo?

Um algoritmo é um conjunto finito e organizado de passos ou instruções que são seguidos para realizar uma tarefa ou resolver um problema específico. É uma sequência lógica de ações que, quando executadas corretamente, levam a um resultado desejado.

Pense em um algoritmo como uma receita de culinária. A receita é um conjunto de instruções claras e precisas sobre o que fazer e em que ordem para preparar um prato específico. Da mesma forma, um algoritmo fornece uma série de etapas bem definidas para realizar uma ação ou chegar a uma conclusão.

Os algoritmos são usados em várias áreas, incluindo ciência da computação, matemática, engenharia e muitas outras disciplinas. Eles são essenciais para a programação de computadores, pois os programadores escrevem algoritmos para criar software que execute tarefas específicas.

Um exemplo simples de algoritmo seria o processo de multiplicação de dois números:

1. Escreva os dois números a serem multiplicados.
2. Multiplique o primeiro número pelo segundo número.

3. O resultado é o produto da multiplicação.

Nesse caso, os passos são claros e precisos, e se você seguir esses passos corretamente, obterá o resultado correto da multiplicação. Isso é um algoritmo em ação.

Começando com a prática

Estrutura básica

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Algoritmo</title>
  <style>
    /* SEU CSS AQUI */
  </style>
</head>
<body>

  <!-- SEU CÓDIGO HTML AQUI -->

  <script>
    // SEU CÓDIGO JAVASCRIPT AQUI
  </script>
</body>
</html>
```

Comandos de saída

Em programação, os comandos de saída são instruções que permitem que um programa exiba informações na tela ou em outro dispositivo de saída, como um arquivo de texto.

Esses comandos são usados para comunicar informações aos usuários ou para verificar o estado do programa durante a execução.

Os comandos de saída variam dependendo da linguagem de programação que você está usando, mas geralmente envolvem a exibição de valores de **variáveis**, mensagens de texto ou resultados de cálculos.

Alguns exemplos de comandos de saída em diferentes linguagens de programação são:

JavaScript

```
console.log("Olá, mundo!"); // Exibe a mensagem no console do navegador
```

Python:

```
print("Olá, mundo!") # Exibe a mensagem na tela
```

C++:

```
#include <iostream>

using namespace std;

int main() {
    cout << "Olá, mundo!" << endl; // Exibe a mensagem na tela
    return 0;
}
```

Java:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Olá, mundo!"); // Exibe a mensagem na tela
    }
}
```

C#:

```
using System;

class Program {
    static void Main() {
        Console.WriteLine("Olá, mundo!"); // Exibe a mensagem na tela
    }
}
```

Em todos esses exemplos, os comandos de saída são usados para mostrar uma mensagem simples na tela ou no console. Eles são uma parte fundamental da programação, pois permitem que os programas interajam com os usuários ou forneçam informações importantes durante a execução.

Saída de dados no JavaScript

Console.Log

No JavaScript, para exibir informações no console do navegador, você pode usar o método **console.log()**. Aqui está um exemplo simples de como usar o comando de saída no console usando JavaScript:

```
console.log("Olá, mundo!"); // Exibe "Olá, mundo!" no console do navegador
```

Você pode abrir o console do navegador (geralmente pressionando F12 ou clicando com o botão direito e selecionando "Inspecionar" e, em seguida, navegando para a guia "Console") para ver a saída resultante.

O **console.log()** é amplamente utilizado para depuração e registro de informações durante o desenvolvimento de aplicativos da web. Ele permite que você visualize variáveis, mensagens e outros dados importantes enquanto seu código está sendo executado no navegador.

Explicando o console.log

console: "Console" se refere a uma interface onde você pode ver mensagens, erros e informações enquanto um programa está sendo executado. No contexto do JavaScript em um navegador, o "console" é uma ferramenta que permite a comunicação entre seu código JavaScript e o ambiente do navegador.

log: "Log" é uma abreviação de "logar" (registrar), que é o ato de registrar informações em um log ou arquivo. No contexto do JavaScript, "log" se refere a exibir uma mensagem ou valor no console para que você possa visualizá-lo durante a execução do programa.

Portanto, **console.log** é um método que faz parte do objeto console no JavaScript, permitindo que você registre informações no console.

Quando você chama **console.log()**, você está instruindo o navegador a exibir uma mensagem ou valor específico no console para que você possa acompanhar o que está acontecendo no seu código enquanto ele é executado. Isso é particularmente útil para depurar erros, verificar o valor de variáveis e acompanhar o fluxo do seu programa.

Exemplos de como utilizar o console.log

```
console.log('Olá mundo!!!');  
console.log('Um exemplo\nde texto\nquebrando linhas');  
console.log('Um exemplo\n\tde texto\n\t\tquebrando linhas');
```

```
console.log('Posso escrever com aspas simples');  
console.log("Posso escrever com aspas duplas")
```

Concatenação, Interpolação e Template String

```
let meuNome = 'Rodrigo';  
let meuSobrenome = 'Dionisio';  
  
console.log(meuNome + ' ' + meuSobrenome);  
console.log(meuNome , meuSobrenome);  
console.log(`${meuNome} ${meuSobrenome}`);
```

Exercícios

Utilizando o comando `'console.log'` faça os exercícios a seguir:

Exercício 1:

Escreva um programa que exiba a mensagem **"Olá, mundo!"** no console.

Exercício 2:

Escreva um programa que exiba o seu nome no console.

Exercício 3:

Crie um programa que exiba a soma de dois números no console.

Exercício 4:

Crie um programa que exiba o dobro de um número no console.

Exercício 5:

Escreva um programa que exiba a idade de um dos seus colegas no console.

Exercício 6:

Escreva um programa que exiba a mensagem **"Estou aprendendo JavaScript!"** 5 vezes no console.

Exercício 7:

Crie um programa que exiba a sequência de números de 1 a 10 no console.

Exercício 8:

Escreva um programa que exiba a mensagem "**Feliz Ano Novo!**" 3 vezes no console.

Exercício 9:

Escreva um programa que exiba o resultado de um número elevado ao cubo.

Exercício 10:

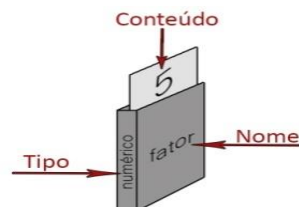
Escreva um programa que exiba o seu nome, sobrenome e idade.

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.

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:

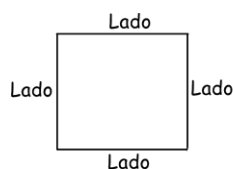
O número digitado foi ..., seu antecessor é ... e seu sucessor é ...

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

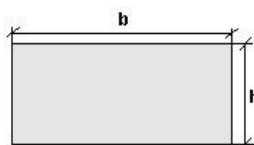


Exercício 19:

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 em anos humanos. Sabendo que a cada ano de vida do cachorro equivale a 7 anos humanos, 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);
```

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 explícito

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:

Exercício 33:

Crie uma aplicação que receba um número e imprima sua raiz quadrada.

SAÍDA:

A raiz quadrada de no número é ...

Exercício 34:

Crie uma aplicação que receba um número e imprima seu valor elevado a 2, elevado a 3, elevado a 4 e elevado a 5.

SAÍDA:

O número digitado foi ... E seu valor elevado a 2 é ..., elevado a 3 é ..., ...

Exercício 35:

Crie uma aplicação que receba um número e imprima sua raiz quadrada e sua raiz cúbica.

SAÍDA:

O número digitado foi ...

Sua raiz cúbica é ...

Sua raiz quadrada é ...

Exercício 36:

Crie uma aplicação que receba com quatro números e imprimir a média ponderada, sabendo-se que os pesos são respectivamente 1, 2, 3 e 4.

Exercício 37:

Crie uma aplicação que receba o raio de uma circunferência e calcule a área e o perímetro do círculo correspondente.

A fórmula para se calcular a área da circunferência é : $A = \pi * \text{raio}^2$

A fórmula para se calcular o perímetro da circunferência é : $P = 2 * \pi * r$

Exercício 38:

Crie uma aplicação que receba os lados A, B e C de um paralelepípedo.

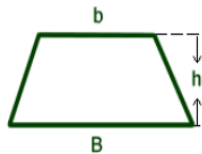
Calcular e imprimir o volume.

- Volume = $A * B * C$

Exercício 39:

Crie uma aplicação para calcular a área de um trapézio qualquer (figura meramente ilustrativa).

$$\text{Área} = ((b + B) * h) / 2$$

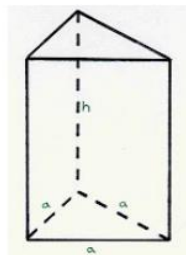


Exercício 40:

Crie uma aplicação que possa calcular o volume de um prisma de base triangular (figura meramente ilustrativa).

Para calcular a área de um triângulo utilize a fórmula “(base * altura)/2”.

Volume = área da base x altura.

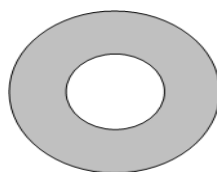


Exercício 41:

Crie uma aplicação que possa calcular a área de uma coroa de forma circular (figura meramente ilustrativa).

$$\text{Área da circunferência} = \pi * \text{raio}^2$$

$$\text{Área} = (\text{Área da circunferência Maior}) - (\text{Área da circunferência menor})$$



Exercício 42:

Crie uma aplicação que possa calcular o volume de um cilindro (figura meramente ilustrativa).

Área da base = área da circunferência

Volume = área da base x altura

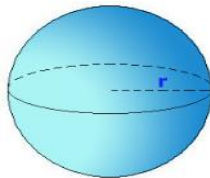


Exercício 43:

Crie uma aplicação para calcular o volume e a área de uma esfera (figura meramente ilustrativa).

Área = $4 * \pi * r^2$

Volume = $(4 * \pi * r^3) / 3$



Operador de atribuição "="

O operador de atribuição = (um único sinal de igual) é um elemento fundamental para as linguagens de programação, incluindo JavaScript.

Sua função é atribuir um valor a uma variável.

Aqui estão os principais pontos a serem considerados sobre o operador de atribuição:

1º) Atribuição de Valor:

O operador = é usado para atribuir um valor a uma variável. O valor à direita do operador é atribuído à variável à esquerda.

Exemplo:

```
let x = 10; // Atribui o valor 10 à variável x
```

2º) Atualização de Variáveis:

O operador de atribuição é comumente usado para atualizar o valor de uma variável. Você pode atribuir um novo valor à variável, substituindo o valor existente.

Exemplo:

```
let y = 5;  
y = y + 3; // Atualiza o valor de y para 8
```

Isso pode ser simplificado usando a notação de atribuição composta:

```
let y = 5;  
y += 3; // Também atualiza o valor de y para 8
```

3º) Expressões à Direita:

A expressão à direita do operador de atribuição é resolvida e seu resultado é atribuído à variável à esquerda.

Exemplo:

```
let a = 5 + 3; // efetua a soma (5 + 3) e atribui o resultado (8) à variável 'a'
```

4º) Atribuição em Cadeia:

Você pode encadear múltiplas atribuições em uma única linha, atribuindo valores a várias variáveis.

Exemplo:

```
let p = 10, q = 20, r = 30; // Atribui 10 a p, 20 a q e 30 a r em uma única linha
```

5º) Operador de Atribuição em Destructuring:

O operador de atribuição também é usado em recursos de JavaScript, como atribuição de desestruturação, para extrair valores de objetos ou arrays e atribuí-los a variáveis.

Exemplo:

```
const person = { name: 'Alice', age: 30 };  
const { name, age } = person; // Atribui 'Alice' a name e 30 a age
```

Em resumo, o operador de atribuição = é essencial para a criação, atualização e manipulação de variáveis em JavaScript e em muitas outras linguagens de programação. Ele permite que você associe valores a variáveis, o que é uma parte fundamental da programação.

Incremento e decremento

Em JavaScript, existem operadores de pré incremento (++variável), pós-incremento (variável++) e operadores de pré-decremento (--variável) e pós-decremento (variável--).

Esses operadores são usados para aumentar ou diminuir o valor de uma variável numérica em 1. No entanto, eles diferem na ordem em que o valor é modificado e quando o valor original é usado em uma expressão.

Vamos explicar cada um deles com exemplos:

Pré-incremento (++variável):

O operador pré-incremento aumenta o valor da variável em 1 e, em seguida, usa o novo valor.

Isso significa que primeiro a variável é incrementada e, em seguida, o valor atualizado é retornado podendo ser utilizado.

Exemplo:

```
let x = 5;
let y = ++x;
console.log(x); // x agora é 6
console.log(y); // y é 6, pois foi incrementado antes de ser atribuído a y
```

Pós-incremento (variável++):

O operador pós-incremento usa o valor atual da variável em uma expressão e, em seguida, aumenta o valor em 1.

Isso significa que primeiro o valor atual é retornado (podendo ser utilizado) e, em seguida, a variável é incrementada.

Exemplo:

```
let x = 5;
let y = x++;
console.log(x); // x agora é 6
console.log(y); // y é 5, pois o valor original de x é usado antes do incremento
```

Pré-decremento (--variável):

O operador pré-decremento diminui o valor da variável em 1 e, em seguida, usa o novo valor.

Isso significa que primeiro a variável é decrementada e, em seguida, o valor atualizado é retornado (podendo ser utilizado).

Exemplo:

```
let x = 5;
let y = --x;
console.log(x); // x agora é 4
console.log(y); // y é 4, pois foi decrementado antes de ser atribuído a y
```

Pós-decremento (variável--):

O operador pós-decremento usa o valor atual da variável em uma expressão e, em seguida, diminui o valor em 1.

Isso significa que primeiro o valor atual é retornado (podendo ser utilizado) e, em seguida, a variável é decrementada.

Exemplo:

```
let x = 5;  
let y = x--;  
console.log(x); // x agora é 4  
console.log(y); // y é 5, pois o valor original de x é usado antes do decremento
```

Lembre-se de que esses operadores podem ser usados para incrementar ou decrementar variáveis numéricas e podem ser úteis em loops, cálculos matemáticos e outras situações em que você precise alterar o valor de uma variável de forma controlada.

Mas por que utilizar o ++ e não o += ou 'n = n+1'

Utilizar o operador ++ em vez de += 1 ou n = n + 1 é uma questão de conveniência e legibilidade do código.

Todos esses métodos podem ser usados para incrementar o valor de uma variável em 1, mas eles têm suas vantagens e desvantagens:

1º) ++ é mais conciso:

O operador ++ é uma forma mais concisa de incrementar uma variável em 1. Isso torna o código mais limpo e mais fácil de ler, especialmente quando você está realizando incrementos simples.

Exemplo:

```
let x = 5;  
x++; // Mais conciso
```

Em vez de:

```
let x = 5;  
x += 1; // Menos conciso  
// ou  
x = x + 1; // Menos conciso
```

2º) ++ é uma operação única:

O operador ++ é uma única operação, enquanto += 1 ou n = n + 1 envolvem duas operações (uma adição e uma atribuição). Em muitos casos, uma única operação pode ser mais eficiente.

3º) ++ é amplamente reconhecido:

O uso de ++ é uma convenção amplamente reconhecida em muitas linguagens de programação, o que torna o código mais legível para outros desenvolvedores.

A maioria dos programadores está familiarizada com esse operador e entende imediatamente seu propósito.

No entanto, a escolha entre essas opções depende do contexto e das preferências de codificação da equipe.

O uso do operador ++ é recomendado quando você deseja realizar um incremento simples de 1. Para incrementos maiores do que 1 ou para situações mais complexas, como incrementar por uma variável diferente de 1, o uso de += ou $n = n + 1$ pode ser mais apropriado.

Incorporando um arquivo JavaScript em uma página HTML

HTML

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JavaScript</title>
</head>
<body>

  <h1>Algoritmo</h1>
  <h3>Lógica de programação</h3>

  <p>
    Fazendo o link entre um arquivo <strong>JavaScript</strong> e a página
    <strong>HTML</strong>.
  </p>

  <script src="arquivoJavaScript.js"></script>
</body>
</html>
```

Exibindo informações no browser

Document Object Model (DOM)

O **DOM**, ou **Modelo de Objeto de Documento** (em inglês, *Document Object Model*), é uma representação da estrutura de um documento HTML ou XML, que pode ser manipulada e acessada por meio de linguagens de programação, como JavaScript.

O **DOM** permite que os desenvolvedores acessem, modifiquem e interajam com os elementos de uma página web de forma dinâmica.

Aqui estão alguns pontos importantes sobre o **DOM** em programação web:

1º) Árvore de Elementos:

O **DOM** representa uma página web como uma árvore de elementos, onde cada elemento (como tags HTML) é representado por um nó (node) na árvore.

A raiz da árvore é o próprio documento (normalmente representado por **document** em JavaScript), e os nós filhos são os elementos da página, como elementos HTML (tags), atributos, texto e assim por diante.

2º) Acesso e Manipulação:

O **DOM** oferece uma interface de programação que permite aos desenvolvedores acessar e manipular elementos HTML de uma página.

Isso significa que você pode usar JavaScript para encontrar elementos, ler ou modificar seus atributos, conteúdo e estilos, bem como adicionar ou remover elementos dinamicamente em uma página web.

3º) Interatividade:

O **DOM** é fundamental para a interatividade em páginas da web. Ele permite que você responda a eventos do usuário, como cliques e pressionamentos de teclas, para alterar a aparência ou o conteúdo da página em tempo real.

4º) Estrutura Hierárquica:

O **DOM** reflete a estrutura hierárquica dos elementos em uma página da web. Isso significa que você pode navegar de um elemento pai para seus filhos e vice-versa, o que é útil ao buscar elementos específicos em uma página complexa.

Aqui está um exemplo simples de manipulação do DOM usando JavaScript para alterar o conteúdo de um elemento HTML:

```
<!DOCTYPE html>
<html>
<body>

<p id="meuParagrafo">Este é um parágrafo.</p>

<script>
    // Acessa o elemento pelo seu id
    const paragrafo = document.getElementById("meuParagrafo");

    // Modifica o conteúdo do parágrafo
    paragrafo.textContent = "Novo conteúdo do parágrafo";
</script>

</body>
</html>
```


O DOM desempenha um papel fundamental no desenvolvimento web, permitindo a criação de páginas interativas e dinâmicas. É uma parte essencial para a manipulação e renderização de conteúdo em resposta às ações dos usuários e eventos do navegador.

document.getElementById

O código **document.getElementById()** é uma função em JavaScript que é usada para obter uma referência a um elemento HTML com base no valor do atributo **id** desse elemento.

Essa função é muito comum e amplamente usada na manipulação do **DOM** em páginas da web.

Aqui está uma explicação mais detalhada do **document.getElementById()**:

- **document:** O objeto **document** representa o documento HTML atual em um navegador da web. Ele é uma parte essencial do DOM e fornece uma interface para interagir com todos os elementos na página.
- **getElementById():** Este é um método do objeto **document**. Ele é usado para buscar um elemento específico com base em seu **id**.
- **id:** O atributo **id** é um identificador exclusivo atribuído a um elemento HTML. Cada elemento deve ter um **id** exclusivo dentro de uma página da web.

Quando você chama **document.getElementById()** e fornece o **id** do elemento que deseja selecionar como argumento, a função retorna uma referência para esse elemento. Isso permite que você acesse e manipule o elemento de várias maneiras, como alterar seu conteúdo, estilo, eventos ou até mesmo remover o elemento.

Aqui está um exemplo de como usar **document.getElementById()**:

Suponha que você tenha o seguinte elemento HTML em sua página:

```
<div id="minhaDiv">Este é um elemento de teste.</div>
```

Você pode acessar esse elemento com JavaScript da seguinte maneira:

```
const elemento = document.getElementById('minhaDiv');
```

Agora, a variável **elemento** contém uma referência ao **<div>** com o **id "minhaDiv"**. Você pode, então, usar essa referência para realizar diversas operações no elemento, como alterar seu conteúdo, estilo ou manipular eventos associados a ele.

Por exemplo:

```
elemento.textContent = 'Texto modificado'; // Altera o conteúdo do elemento
elemento.style.backgroundColor = 'blue'; // Altera o estilo do elemento
elemento.addEventListener('click', function() {
```

```
    alert('Clicou na div!');  
  }); // Adiciona um evento de clique ao elemento
```

Essa é uma maneira fundamental de interagir com elementos HTML em uma página da web usando JavaScript e é amplamente utilizada no desenvolvimento web.

innerHTML

O **innerHTML** é uma propriedade em JavaScript que permite acessar e modificar o conteúdo HTML de um elemento do DOM.

Essa propriedade fornece uma maneira conveniente de manipular o conteúdo interno de um elemento HTML, incluindo texto e elementos filhos. Aqui está uma explicação sobre o **innerHTML**:

Acesso ao Conteúdo HTML:

A propriedade **innerHTML** permite que você acesse o conteúdo HTML de um elemento, incluindo seu texto e outros elementos HTML que estão dentro dele.

Leitura do Conteúdo:

Você pode ler o conteúdo atual de um elemento usando **innerHTML**. Por exemplo:

```
const elemento = document.getElementById('meuElemento');  
const conteudoHTML = elemento.innerHTML;  
console.log(conteudoHTML);
```

Isso retornará o conteúdo HTML do elemento com o ID "**meuElemento**."

Modificação do Conteúdo:

Você também pode usar **innerHTML** para modificar o conteúdo de um elemento, substituindo-o por uma nova string HTML. Isso pode ser útil para adicionar, atualizar ou remover elementos dentro de um elemento pai.

Exemplo de modificação do conteúdo:

```
const elemento = document.getElementById('meuElemento');  
elemento.innerHTML = '<p>Novo conteúdo HTML</p>';
```

Isso substituirá todo o conteúdo dentro do elemento com o novo conteúdo HTML.

Cuidados de Segurança:

É importante ter cuidado ao usar **innerHTML**, pois ele pode abrir brechas de segurança se o conteúdo inserido não for confiável. Isso pode resultar em ataques de injeção de código. É recomendável usar outras técnicas, como **textContent**, quando você precisa manipular conteúdo de texto simples e confiável.

Aninhamento de Elementos:

O **innerHTML** permite o aninhamento de elementos HTML. Isso significa que você pode inserir elementos HTML dentro de outros elementos usando essa propriedade.

Exemplo de aninhamento de elementos:

```
const elemento = document.getElementById('meuElemento');
elemento.innerHTML = '<p>Parágrafo <span>aninhado</span></p>';
```

Nesse caso, um parágrafo (<p>) contém um elemento span aninhado.

O uso de **innerHTML** pode ser muito conveniente para tarefas específicas de manipulação de conteúdo HTML, mas lembre-se de usar com responsabilidade e cuidado, especialmente quando a entrada do usuário ou dados externos não confiáveis estiverem envolvidos, para evitar vulnerabilidades de segurança. Em muitos casos, dependendo da tarefa, pode ser mais seguro usar métodos como **textContent** para manipular apenas o texto do elemento.

Entrada de dados

Prompt

O comando **prompt** em JavaScript é uma função que permite que você exiba uma caixa de diálogo de entrada ao usuário em um navegador da web. Essa caixa de diálogo geralmente é usada para coletar dados inseridos pelo usuário, como texto, números ou informações diversas.

A sintaxe básica da função **prompt** é a seguinte:

```
let resultado = prompt("Texto da mensagem para o usuário", "Valor Padrão");
```

Aqui estão os componentes principais:

Texto da mensagem para o usuário: Esta é a mensagem ou pergunta que será exibida para o usuário na caixa de diálogo. É uma string que fornece contexto sobre o que o usuário deve inserir.

Valor Padrão (opcional): Este é um valor padrão que pode ser pré-preenchido na caixa de entrada. O usuário pode aceitar esse valor padrão ou substituí-lo.

O prompt retorna o valor inserido pelo usuário como uma string. Se o usuário pressionar "Cancelar" na caixa de diálogo, o resultado será null.

Exemplo de uso do prompt:

```
let nome = prompt("Digite seu nome:");
if (nome !== null) {
  console.log("Olá, " + nome + "!");
} else {
  console.log("Você cancelou a entrada.");
}
```

Neste exemplo, uma caixa de diálogo será exibida solicitando que o usuário insira seu nome. Se o usuário inserir um nome e clicar em "OK", o nome será armazenado na variável nome e saudado na próxima linha. Se o usuário clicar em "Cancelar", o prompt retorna null, e uma mensagem de cancelamento será exibida.

Tenha em mente que, como o **prompt** retorna uma string, você pode precisar fazer conversões de tipo (casting) se desejar que o valor inserido seja tratado como um número, por exemplo. Certifique-se também de validar os dados inseridos pelo usuário para garantir que sejam adequados ao que você espera em seu código.

Exercícios:

Exercício 44: Cálculo de Idade em Dias

Você deve calcular a idade de uma pessoa em dias. Insira sua idade em anos e, em seguida, calcule e exiba a idade em dias. Lembrando que um ano tem 365 dias. (Exiba os dados no browser e no console).

Exercício 45:

Antes do racionamento de energia ser decretado, quase ninguém falava em quilowatts, mas agora, todos incorporaram essa palavra em seu vocabulário.

Sabendo-se que 100 quilowatts de energia custa um sétimo do salário mínimo, faça uma aplicação que receba o valor do salário mínimo e a quantidade de quilowatts gasta por uma residência.

Calcule e exiba no browser:

- O valor em reais de cada quilowatt.
- O valor em reais a ser pago.
- O novo valor a ser pago por essa residência com um desconto de 10%.

Exercício 46:

Crie uma aplicação que receba a temperatura em graus Celsius e a exiba convertida em graus Fahrenheit.

A fórmula de conversão é: $F = C * (9.0/5.0) + 32.0$, sendo F a temperatura em Fahrenheit e C a temperatura em Celsius. (Exiba os dados no browser e no console).

Exercício 47:

Crie uma aplicação que receba o salário de um funcionário e exiba o valor deste salário, calcular e mostrar seu novo salário, sabendo que ele recebeu um aumento de 25%. (Exiba os dados no browser e no console).

Exercício 48:

Crie uma aplicação que receba a altura do degrau de uma escada (cm) e a altura que o usuário deseja alcançar subindo a escada (cm). Calcule e mostre quantos degraus o usuário deverá subir para atingir seu objetivo, sem se preocupar com a altura do usuário. (Exiba os dados no browser e no console).

Exercício 49: IMC

Crie um programa que calcule o Índice de Massa Corporal (IMC) de uma pessoa. O programa deve receber o peso em quilogramas e a altura em metros. O programa deve exibir o IMC calculado. (Exiba os dados no browser e no console).

A fórmula para calcular o IMC é:

$$\text{IMC} = \text{peso (em quilogramas)} / (\text{altura (em metros)} * \text{altura (em metros)})$$

Exercício 50: Calculando o Preço Total de Compras

Crie um programa para calcular o preço total de compras em uma loja. O programa deve receber o preço de três produtos diferentes. O programa deve somar os preços e exibir o total a ser pago sem desconto, com desconto de 10%, desconto de 20%, desconto de 30% e desconto de 50%. (Exiba os dados no browser e no console).

Exercício 51: Calculando a Média de Notas

Crie um programa para calcular a média aritmética de cinco notas. O programa deve receber as cinco notas, deve calcular a média aritmética e exibir o resultado no browser.

Exercício 52: Cálculo de Gorjeta

Faça um programa que calcule a gorjeta a ser deixada em um restaurante. O programa deve receber o valor da conta e a porcentagem de gorjeta desejada. O programa deve calcular e exibir no browser o valor da gorjeta e o total a ser pago.

Exercício 53: Aumento de População

Crie um programa que calcule o aumento da população ao longo de um ano. O programa deve receber a (quantidade) população inicial, a taxa de crescimento anual (em porcentagem). O programa deve calcular e exibir no browser a população futura após o período especificado.

Exercício 54: Cálculo de Idade em Semanas

Crie um programa que receba a idade de uma pessoa em anos. Calcule e exiba quantas semanas de vida essa pessoa tem aproximadamente (considerando 52 semanas por ano).

Estruturas condicionais

As **estruturas condicionais** são elementos fundamentais na programação, pois elas permitem que um programa tome decisões com base em condições específicas. Elas permitem que você controle o fluxo de execução do código, determinando se certas instruções devem ser executadas ou não, dependendo de sua condição (verdadeira ou falsa).

As estruturas condicionais permitem que você possa verificar se uma determinada expressão lógica (condição) é verdadeira ou não.

Se a expressão for verdadeira ela executa um determinado trecho do código e em caso contrário poderá ser executado outro trecho de código ou simplesmente não executar nenhum trecho de código.

Uma expressão condicional sempre irá retornar um valor booleano, em outras palavras ela sempre irá retornar um valor **verdadeiro** ou **falso**.

As estruturas condicionais são importantes por várias razões:

1. **Tomada de Decisões:** As estruturas condicionais permitem que os programas tomem decisões lógicas com base em dados ou entradas do usuário. Isso é essencial para criar programas que se adaptem dinamicamente às circunstâncias.

2. **Controle de Fluxo:** Elas fornecem controle de fluxo, permitindo que você especifique diferentes caminhos de execução do programa com base nas condições. Isso é útil para criar programas que respondam de maneira apropriada a diferentes cenários.

3. **Eficiência:** As estruturas condicionais também podem aumentar a eficiência do código, evitando a execução de instruções desnecessárias. Se uma condição não for satisfeita, as instruções relacionadas a ela podem ser ignoradas.

4. **Personalização:** Com as estruturas condicionais, você pode personalizar a interação do seu programa com o usuário ou com os dados, tornando-o mais flexível e útil.

Existem vários tipos de estruturas condicionais em programação, incluindo:

- **IF:** O "if" é a estrutura condicional mais básica, que permite que um bloco de código seja executado somente se uma condição for verdadeira.

- **ELSE:** O "else" é obrigatoriamente usado em conjunto com o "if" e permite a execução de um bloco de código alternativo caso a condição do "if" seja falsa.
- **ELSE IF (ou ELIF):** O "else if" (ou "elif" em algumas linguagens) é usado para verificar múltiplas condições sequencialmente e executar o bloco de código associado à primeira condição verdadeira encontrada.
- **Switch/Case (ou Seleção por Caso):** Esta estrutura é usada para lidar com múltiplas condições diferentes e direcionar a execução do código para diferentes blocos de instruções com base no valor de uma variável ou expressão.

Em resumo, as estruturas condicionais são fundamentais para criar programas que podem tomar decisões lógicas e se adaptar a diferentes situações, tornando-os mais poderosos e flexíveis. Elas desempenham um papel crucial na lógica de programação e são uma habilidade essencial para qualquer desenvolvedor de software.

Operador	Significado
==	Igual a
!=	Diferente de
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a
===	Idêntico
!==	Não idêntico

Estrutura condicional IF..., IF... ELSE e IF... ELSE IF... ELSE

Estrutura Condicional "IF"

A estrutura condicional "if" é usada para executar um bloco de código apenas se uma determinada condição for verdadeira. Aqui está a sintaxe básica:

```
if (condicao) {  
    // Código a ser executado se a condição for verdadeira  
}
```

Exemplo:

```
let idade = 20;  
if (idade >= 18) {  
    console.log("Você é maior de idade.");  
}
```

Neste exemplo, o código dentro do bloco "if" só será executado se a variável "idade" for maior ou igual a 18.

Estrutura Condicional "IF/ELSE"

A estrutura condicional "**if/else**" é usada quando você deseja executar um bloco de código se uma condição for verdadeira e outro bloco de código se a condição for falsa. Aqui está a sintaxe básica:

```
if (condicao) {  
    // Código a ser executado se a condição for verdadeira  
} else {  
    // Código a ser executado se a condição for falsa  
}
```

Exemplo:

```
let idade = 15;  
if (idade >= 18) {  
    console.log("Você é maior de idade.");  
} else {  
    console.log("Você é menor de idade.");  
}
```

Neste exemplo, se a idade for maior ou igual a 18, a mensagem "Você é maior de idade." será exibida. Caso contrário, a mensagem "Você é menor de idade." será exibida.

IF... ELSE IF... ELSE

A estrutura condicional "**if...else if...else**" é usada quando você precisa avaliar várias condições sequencialmente e executar diferentes blocos de código com base na primeira condição verdadeira encontrada. Isso permite criar uma cadeia de verificações condicionais mais complexas. Aqui está a sintaxe básica:

```
if (condicao1) {  
    // Código a ser executado se a condição1 for verdadeira  
} else if (condicao2) {  
    // Código a ser executado se a condição2 for verdadeira  
} else {  
    // Código a ser executado se nenhuma das condições anteriores for verdadeira  
}
```

Vamos ver um exemplo em JavaScript para ilustrar como isso funciona:

```
let diaSemana = "quarta";  
  
if (diaSemana === "segunda") {  
    console.log("É segunda-feira.");  
} else if (diaSemana === "terça") {  
    console.log("É terça-feira.");  
} else if (diaSemana === "quarta") {  
    console.log("É quarta-feira.");  
}
```



```
} else {  
  console.log("É um dia diferente da semana.");  
}
```

Neste exemplo, a variável **diaSemana** é avaliada em várias condições sequencialmente. Se **diaSemana** for igual a "segunda", será exibida a mensagem "É segunda-feira." Se for igual a "terça", a mensagem será "É terça-feira." Se for igual a "quarta", a mensagem será "É quarta-feira." Se nenhuma das condições anteriores for verdadeira, a mensagem "É um dia diferente da semana." será exibida.

Esta estrutura condicional é útil quando você tem várias condições para verificar e deseja executar um bloco de código correspondente à primeira condição verdadeira encontrada. Tenha em mente que o código dentro de um bloco "else if" só será executado se a condição associada ao "if" ou ao "else if" correspondente for verdadeira. Se nenhuma das condições for verdadeira, o bloco "else" será executado (se presente).

Operador Ternário

O **operador ternário** em JavaScript é uma forma concisa de escrever uma estrutura condicional "if...else" em uma única linha. Ele é frequentemente usado para atribuir um valor a uma variável com base em uma condição. A sintaxe básica do operador ternário é a seguinte:

```
condicao ? valorSeVerdadeiro : valorSeFalso
```

Onde:

- **condicao**: A expressão condicional que será avaliada. Se esta condição for verdadeira, o valor antes dos dois pontos (:) será retornado. Caso contrário, o valor após os dois pontos (:) será retornado.
- **valorSeVerdadeiro**: O valor a ser retornado se a condição for verdadeira.
- **valorSeFalso**: O valor a ser retornado se a condição for falsa.

Aqui está um exemplo simples:

```
let idade = 20;  
let mensagem = idade >= 18 ? "Maior de idade" : "Menor de idade";  
  
console.log(mensagem); // Isso irá imprimir "Maior de idade"
```

Neste exemplo, a condição **idade >= 18** é avaliada. Se for verdadeira, a mensagem "Maior de idade" será atribuída à variável **mensagem**; caso contrário, a mensagem "Menor de idade" será atribuída. Como **idade** é 20, a primeira condição é verdadeira e, portanto, "Maior de idade" é atribuído à variável **mensagem**.

Outros exemplos:

```
let nota = 5.00;
let texto = (nota >= 6.00) ? "Aprovado" : "Reprovado";
console.log(texto);

nota = 9.99;
console.log((nota >= 6.00) ? "Aprovado" : "Reprovado");
```

O operador ternário é útil quando você deseja atribuir um valor com base em uma condição de maneira sucinta e legível. No entanto, ele deve ser usado com moderação para manter o código legível, já que pode se tornar confuso quando usado em expressões muito complexas.

Operadores lógicos

Operador lógico	Significado
&& (and)	Se duas expressões condicionais forem verdadeiras, o resultado é verdadeiro
(or)	Se qualquer expressão condicional é verdadeira, o resultado é verdadeiro
! (not)	Se a expressão condicional for falsa, o resultado é verdadeiro. Se a expressão condicional for verdadeira, o resultado é falso

Em JavaScript, os operadores lógicos **&&** (E lógico), **||** (OU lógico) e **!** (NÃO lógico) são usados para realizar operações lógicas em valores booleanos (verdadeiro ou falso). Eles são frequentemente usados em estruturas condicionais para criar expressões lógicas mais complexas. Vamos explicar cada um deles:

1. Operador && (E lógico):

O operador **&&** realiza uma operação de **E lógico** entre dois valores booleanos. Ele retorna **true** se ambos os operandos forem verdadeiros e **false** se pelo menos um deles for falso.

Exemplo:

```
let a = true;
let b = false;

let resultado = a && b; // resultado é igual a false
```

Neste exemplo, **resultado** será *false* porque ambos **a** e **b** precisam ser verdadeiros para que a expressão seja verdadeira.

Outro exemplo com o operador lógico &&:

Suponha que você queira verificar se uma pessoa é elegível para votar em uma eleição com base em sua idade e cidadania. Para votar, a pessoa deve ser maior de idade (18 anos ou mais) e ser cidadã do país. Você pode usar o operador && para verificar ambas as condições:

```
let idade = 20;
let eCidadao = true;

if (idade >= 18 && eCidadao == true) {
  console.log("Você é elegível para votar.");
} else {
  console.log("Você não é elegível para votar.");
}
```

Neste exemplo, o código dentro do bloco "if" só será executado se ambas as condições, idade >= 18 e eCidadao == true, forem verdadeiras.

2. Operador || (OU lógico):

O operador || realiza uma operação de **OU lógico** entre dois valores booleanos. Ele retorna **true** se pelo menos um dos operandos for verdadeiro e false somente se ambos os operandos forem falsos.

Exemplo:

```
let a = true;
let b = false;

let resultado = a || b; // resultado é igual a true
```

Neste exemplo, resultado será **true** porque pelo menos um dos operandos (a) é verdadeiro.

Outro exemplo com o operador lógico ||:

Agora, suponha que você deseje verificar se um usuário tem acesso a um conteúdo premium com base em sua assinatura mensal ativa ou em uma assinatura anual ativa. Se qualquer uma das condições for verdadeira, o usuário terá acesso ao conteúdo premium:

```
let assinaturaMensualAtiva = false;
let assinaturaAnualAtiva = true;

if (assinaturaMensualAtiva || assinaturaAnualAtiva) {
  console.log("Você tem acesso ao conteúdo premium.");
}
```

```
    } else {  
        console.log("Você não tem acesso ao conteúdo premium.");  
    }
```

Neste exemplo, o código dentro do bloco "if" será executado se pelo menos uma das condições, **assinaturaMensalAtiva** ou **assinaturaAnualAtiva**, for verdadeira.

3. Operador ! (NÃO lógico):

O operador ! (NÃO lógico) é usado para inverter o valor de uma expressão booleana. Ou seja, ele retorna true se a expressão for falsa e false se a expressão for verdadeira.

Exemplo:

```
let a = true;  
let resultado = !a; // resultado é igual a false
```

Neste exemplo, resultado será **false** porque **!a** inverte o valor de 'a', que era **true**.

Outro exemplo com o operador lógico !:

Imagine que você deseja verificar se um usuário não está banido de um fórum. Você pode usar o operador '!' para inverter o valor de uma variável que indica se o usuário está banido:

```
let usuarioBanido = false;  
  
if (!usuarioBanido) {  
    console.log("Você pode postar no fórum.");  
} else {  
    console.log("Você está banido do fórum.");  
}
```

Neste exemplo, o operador '!' é usado para inverter o valor de **usuarioBanido**, para que o código dentro do bloco "if" seja executado quando o usuário não estiver banido.

Os operadores lógicos são fundamentais para criar condições mais complexas em estruturas condicionais, onde você pode combinar várias expressões booleanas para tomar decisões com base em critérios mais elaborados. Eles também são úteis em outras situações em que a lógica booleana é necessária, como validação de formulários, controle de loops e etc.

Exercícios:

Exercício 55:

Crie uma aplicação que receba quatro notas de um aluno e imprima a média ponderada destas notas, sabendo-se que os pesos são respectivamente 3, 5, 6 e 6.

Imprima também se o aluno está aprovado ou reprovado sabendo-se que para ser aprovado a média deve ser maior que 6,00.

Exercício 56:

Crie uma aplicação que receba três números e imprimir a média aritmética. Imprima também se o aluno está aprovado ou reprovado sabendo-se que para ser aprovado a média deve ser maior ou igual à 7,50.

Exercício 57:

Crie uma aplicação que receba um número qualquer.

Se o número for par o algoritmo deve imprimir seu valor e seu valor elevado ao quadrado.

Se o número for ímpar o algoritmo deve imprimir seu valor e seu valor elevado ao cubo.

Exercício 58:

Crie uma aplicação que calcule e imprima a área e a hipotenusa de um triângulo retângulo.

Observação: Os valores devem ser positivos, caso contrário uma mensagem de erro deve ser impressa para o usuário.

Exercício 59:

Crie uma aplicação que receba um valor qualquer e imprima se o valor digitado é “Par” ou “Ímpar”.

Atenção: os números devem ser maiores que zero, caso contrário uma mensagem de erro deve ser impressa para o usuário.

Exercício 60:

Crie uma aplicação que receba um número qualquer positivo e maior que zero.

Se o valor do número for par o algoritmo deverá imprimir a seguinte mensagem:

- “O número digitado foi ____.”
- “Seu valor elevado ao quadrado é ____.”
- “Seu valor elevado ao cubo é ____.”
- “Seu valor elevado a sétima é ____.”

Se o valor for ímpar o algoritmo deverá imprimir a seguinte mensagem:

- “O número digitado foi ____.”
- “Sua raiz quadrada é ____.”
- “Sua raiz cúbica é ____.”
- “Sua raiz a sétima é ____.”

Observação: Se o usuário digitar um valor inválido o algoritmo deverá emitir uma mensagem de erro.

Exercício 61:

Crie uma aplicação que receba três números.

Exibir os três números informando se eles são positivos, negativos ou nulos (considere o número nulo se ele for igual a zero).

Informar o maior número.

Exercício 62:

Crie uma aplicação que receba dois valores numéricos e efetue sua adição.

Caso o resultado da adição seja maior que 10, exibir os números digitados, o valor da adição e a raiz cúbica da adição.

Caso contrário exibir somente os valores digitados e o valor da adição.

Exercício 63:

Crie uma aplicação que faça a leitura de três valores e apresente como resultado final a soma dos quadrados dos três valores lidos. Apresentar também se a soma é um número par ou ímpar.

Exercícios de análise:

Você deve ler e interpretar os exercícios, depois selecione a alternativa correta.

Observação:

- Não execute o código
- Olhe a resposta somente após a execução do exercício

Exercício 64:

Analise o código e assinale a alternativa correta:

```
let numero = 5;

if (numero % 2 = 0) {
  console.log("Par");
} else {
  console.log("Ímpar");
}
```

Aqui estão as alternativas:

- A) O código está correto. Ele vai imprimir "Ímpar" quando **numero** for ímpar e "Par" quando **numero** for par.
- B) O operador = deve ser substituído por === na linha if (numero % 2 = 0).
- C) A variável **numero** deve ser declarada com **var** em vez de **let**.
- D) A variável **numero** deve ser inicializada com um número decimal, como let numero = 2.5, para testar corretamente se o código funciona.
- E) A variável **numero** deve ser inicializada com uma string, como let numero = "5", para testar corretamente se o código funciona.

Exercício 65:

Analise o código e assinale a alternativa correta:

```
let numero = -7;

if (numero > 0) {
  console.log("Positivo");
} else {
  console.log("Negativo");
}
```

Aqui estão as alternativas:

- A) O código está correto. Ele vai imprimir "Negativo" quando numero for negativo e "Positivo" quando numero for positivo.
- B) O operador > deve ser substituído por === na linha if (numero > 0).
- C) A variável numero deve ser declarada com var em vez de let.
- D) A variável numero deve ser inicializada com um número decimal, como let numero = -3.5, para testar corretamente se o código funciona.
- E) A variável numero deve ser inicializada com uma string, como let numero = "-7", para testar corretamente se o código funciona.

Exercício 66:

Analise o código e assinale a alternativa correta:

Você está criando um programa para calcular se uma pessoa tem um índice de massa corporal (IMC) saudável com base em seu peso e altura. Abaixo está o código do programa que você escreveu diretamente no programa principal. Qual é a alternativa correta para determinar se o código funciona corretamente?

```
let peso = 70;
let altura = 1.75;
let imc = peso / (altura * altura);

if (imc >= 18.5 && imc <= 24.9) {
  console.log("IMC saudável");
} else {
  console.log("IMC não saudável");
}
```

Aqui estão as alternativas:

- A) O código está correto. Ele vai imprimir "IMC saudável" quando o IMC estiver entre 18.5 e 24.9 (inclusive), caso contrário, imprimirá "IMC não saudável".
- B) O operador <= deve ser substituído por < na linha if (imc >= 18.5 && imc <= 24.9).
- C) A variável altura deve ser inicializada com um número inteiro, como let altura = 175, para testar corretamente se o código funciona.

D) Todas as alternativas estão erradas.

E) A variável peso deve ser inicializada com uma string, como let peso = "70", para testar corretamente se o código funciona.

Exercício 67:

Você está criando um programa para contar quantas vezes uma pessoa clica em um botão em um site. O programa deve registrar o número atual de cliques e, em seguida, incrementar esse número sempre que o botão for clicado. Abaixo está o código que você escreveu:

```
let numCliques = 0;

// Simulação de clique no botão
numCliques++;

// Simulação de mais cliques no botão
numCliques += 2;

console.log("Número de cliques: " + numCliques);
```

Qual será o valor impresso no console após a execução deste código?

- A) O valor impresso será "Número de cliques: 0".
- B) O valor impresso será "Número de cliques: 1".
- C) O valor impresso será "Número de cliques: 2".
- D) O valor impresso será "Número de cliques: 3".
- E) O valor impresso será "Número de cliques: 4".

Exercício 68:

Você está criando um programa para registrar a quantidade de itens vendidos em uma loja. O programa deve inicializar o contador de itens vendidos como zero e, em seguida, incrementá-lo sempre que um item for vendido. Abaixo está o código que você escreveu:

```
let itensVendidos = 0;

// Simulação de venda
itensVendidos += 3;

// Simulação de venda
itensVendidos += 2;

console.log("Itens vendidos: " + itensVendidos);
```

Qual será o valor impresso no console após a execução deste código?

- A) O valor impresso será "Itens vendidos: 0".
- B) O valor impresso será "Itens vendidos: 1".
- C) O valor impresso será "Itens vendidos: 3".
- D) O valor impresso será "Itens vendidos: 5".
- E) O valor impresso será "Itens vendidos: 6".

Exercício 69:

Você está desenvolvendo um programa que controla o estoque de produtos em uma loja. O programa inicializa a quantidade de um produto em estoque e, em seguida, decrementa essa quantidade sempre que um produto é vendido. Abaixo está o código que você escreveu:

```
let estoqueProdutoA = 10;  
estoqueProdutoA -= 3;  
estoqueProdutoA -= 2;  
  
console.log("Quantidade de Produto A em estoque: " + estoqueProdutoA);
```

Qual será o valor impresso no console após a execução deste código?

- A) O valor impresso será "Quantidade de Produto A em estoque: 10".
- B) O valor impresso será "Quantidade de Produto A em estoque: 7".
- C) O valor impresso será "Quantidade de Produto A em estoque: 5".
- D) O valor impresso será "Quantidade de Produto A em estoque: 3".
- E) O valor impresso será "Quantidade de Produto A em estoque: 0".

Exercício 70:

Você está desenvolvendo um programa que monitora a contagem de pessoas em uma sala de conferências. O programa começa com uma contagem inicial e, em seguida, intercala a entrada e a saída de pessoas usando os operadores de incremento ++, += e decremento --, -=. Abaixo está o código que você escreveu:

```
let contagemPessoas = 0;  
  
contagemPessoas += 3;  
contagemPessoas--;  
contagemPessoas += 2;  
contagemPessoas--;  
  
console.log("Número de pessoas na sala: " + contagemPessoas);
```

Qual será o valor impresso no console após a execução deste código?

- A) O valor impresso será "Número de pessoas na sala: 0".
- B) O valor impresso será "Número de pessoas na sala: 2".
- C) O valor impresso será "Número de pessoas na sala: 3".
- D) O valor impresso será "Número de pessoas na sala: 4".
- E) O valor impresso será "Número de pessoas na sala: 5".
- F) Todas as alternativas estão erradas.

Estrutura de repetição

Uma estrutura de repetição, em programação, é uma construção que permite que um conjunto de instruções seja executado repetidamente enquanto uma condição específica for atendida.

Essa condição é chamada de "**condição de parada**" e é verificada antes de cada execução do conjunto de instruções. Quando a condição não é mais verdadeira, a repetição para.

As estruturas de repetição são fundamentais para automatizar tarefas que precisam ser executadas várias vezes sem a necessidade de escrever o mesmo código repetidamente. Elas permitem que você crie programas mais eficientes e concisos.

Estrutura de repetição FOR

Na estrutura de repetição FOR um conjunto de instruções é executado enquanto uma condição for satisfeita

Sintaxe:

```
for (inicializacao; condicao; incremento) {  
    // conjunto de instruções  
}
```

Estrutura de repetição WHILE

Na estrutura de repetição **WHILE** a condição é verificada antes da execução do conjunto de instruções. Se a condição já for falsa desde o início, o conjunto de instruções pode nunca ser executado.

Sintaxe:

```
while (condicao) {  
    // conjunto de instruções  
}
```

Estrutura de repetição DO-WHILE

A estrutura de repetição **DO-WHILE** é semelhante ao **WHILE**, mas com uma diferença importante: a condição é verificada após a execução do conjunto de instruções.

Isso significa que o conjunto de instruções sempre será executado pelo menos uma vez, mesmo que a condição seja inicialmente falsa.

Sintaxe:

```
do {  
    // Conjunto de instruções  
} while (condicao);
```

O conjunto de instruções é executado primeiro, independentemente da condição. Após a execução do conjunto de instruções, a condição é verificada. Se a condição for verdadeira, o conjunto de instruções será executado novamente. Isso continuará até que a condição se torne falsa. Se a condição for falsa desde o início, o conjunto de instruções ainda será executado uma vez.

A estrutura **DO-WHILE** é útil quando você deseja garantir que um conjunto de instruções seja executado pelo menos uma vez, independentemente da condição inicial.

As estruturas de repetição são essenciais para realizar tarefas como processar elementos em uma lista (array), ler dados de um arquivo, criar jogos, realizar cálculos complexos e muito mais. Elas são uma parte fundamental da programação e permitem que os programas realizem ações repetitivas de maneira eficiente.

Estrutura de repetição FOR com mais detalhes

A estrutura de repetição **FOR** é uma das estruturas mais comuns em programação e é usada para executar um conjunto de instruções repetidamente por um número específico de vezes. A estrutura **FOR** é especialmente útil quando você sabe antecipadamente quantas vezes deseja repetir um bloco de código.

Aqui está a sintaxe básica da estrutura "for" em JavaScript:

```
for (inicializacao; condicao; incremento ou decremento) {  
    // conjunto de instruções  
}
```

Onde:

- **inicialização:** É onde você define uma variável e atribui um valor inicial. Essa parte é executada apenas uma vez, no início do loop.

- **condição:** É uma expressão que é avaliada antes de cada iteração do loop. Se a condição for verdadeira, o conjunto de instruções dentro do loop será executado. Se for falsa, o loop será interrompido.
- **Incremento ou decremento:** É onde você atualiza a variável definida na inicialização. Geralmente, isso envolve incrementar ou decrementar o valor da variável. Esta parte é executada após cada iteração da estrutura.
- **Conjunto de instruções:** É o bloco de código que será executado repetidamente enquanto a condição for verdadeira.

Exemplo:

```
for (let i = 1; i <= 5; i++) {  
  console.log(i);  
}
```

Neste exemplo:

- A variável **i** é inicializada com o valor 1.
- A condição verifica se **i** é menor ou igual a 5. Se for verdadeira, o loop continua.
- Após cada iteração, **i** é incrementado em 1.
- O conjunto de instruções dentro do loop exibe o valor de **i**.

O resultado deste loop será:

```
1  
2  
3  
4  
5
```

A estrutura **FOR** é uma estrutura de repetição muito versátil e pode ser usada em várias situações, como percorrer elementos de uma lista (array), gerar sequências de números, processar dados em lotes e muito mais. Eles fornecem um controle preciso sobre o número de iterações, tornando-os uma ferramenta fundamental na programação.

Exercícios

Exercício 71:

Crie uma aplicação para imprimir a soma de todos os números de 0 à 100.

Exercício 72:

Escreva um programa que exiba todos os números pares de 1 a 50.

Exercício 73:

Escreva um programa que calcule o fatorial de um número fornecido pelo usuário.

O fatorial de um número é um conceito matemático usado para calcular o produto de todos os números inteiros positivos de 1 até esse número. É denotado pelo símbolo "!".

O fatorial de um número "n" é calculado da seguinte forma:

$$n! = n \times (n - 1) \times (n - 2) \times \dots \times 2 \times 1$$

Por exemplo:

- O fatorial de 5 é calculado como $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$.
- O fatorial de 4 é calculado como $4! = 4 \times 3 \times 2 \times 1 = 24$.
- O fatorial de 0 (zero) é por definição 1.

Exercício 74:

Escreva um programa que gere uma sequência de números a partir de um número inicial fornecido pelo usuário e exiba os primeiros 10 números dessa sequência.

Exemplo:

Número fornecido: 5

Sequência: 5, 10, 15, 20, 25, 30, 35, 40, 45, 50

Exercício 75:

Escreva um programa que exiba a tabuada de um número fornecido pelo usuário.

Exercício 76:

Escreva um programa que exiba os números de 1 a 20. Para cada número, verifique se ele é par ou ímpar e exiba essa informação.

Exercício 77:

Escreva um programa que solicite ao usuário que insira 5 notas e, em seguida, calcule e exiba a média das notas.

Exercício 78:

Escreva um programa que solicite ao usuário que insira 10 números.

Este programa deve contar quantos números são positivos e maiores que zero, contar quantos números são negativos e exiba os resultados.

Exercício 79:

Crie uma aplicação que receba cinco números e imprima o quadrado de cada número.

Exercício 80:

Crie uma aplicação que receba o nome e duas notas de um aluno.

As notas vão de zero a dez. O algoritmo deve imprimir o nome do aluno, suas notas e sua média.

- Se a nota for maior que 7 – imprimir “Aprovado”.
- Se a nota for menor que 5 – imprimir “Retido”.
- Caso contrário - imprimir “Recuperação”.
- Se as notas não estiverem no intervalo estabelecido o algoritmo deve emitir uma mensagem de erro.

Exercício 81:

Crie uma aplicação que receba um número qualquer.

Se o número for positivo, imprimir o número digitado e sua raiz quadrada.

Se o número for negativo, imprimir o número digitado e seu valor elevado ao quadrado.

Estrutura condicional SWITCH

A estrutura condicional **switch** é utilizada para realizar múltiplas comparações de uma expressão. Ela é geralmente usada quando temos várias condições a serem verificadas em relação ao valor de uma única variável.

A sintaxe básica é a seguinte:

```
switch (expressão) {  
  
    case valor1:  
        // código a ser executado se expressão for igual a valor1  
        break;  
  
    case valor2:  
        // código a ser executado se expressão for igual a valor2  
        break;  
  
    // mais casos podem ser adicionados conforme necessário  
    default:  
        // código a ser executado se nenhum caso corresponder a expressão  
  
}
```

Aqui temos mais alguns exemplos simples com **switch**:

Exemplo:

```
let diaDaSemana = 3;  
  
switch (diaDaSemana) {  
  
    case 1:  
        console.log("Domingo");  
        break;
```

```
case 2:
  console.log("Segunda-feira");
  break;

case 3:
  console.log("Terça-feira");
  break;

// mais casos podem ser adicionados conforme necessário
default:
  console.log("Dia inválido");

}
```

Exemplo:

```
let mes = 1;

switch( mes ) {
  case 1:
    console.log("Janeiro")
    break;
  case 2:
    console.log("Fevereiro")
    break;
  case 3:
    console.log("Macço")
    break;
  default:
    console.log("Mês não encontrado")
    break;
}
```

Exemplo:

```
let mesTexto = 'fev';

switch( mesTexto ) {
  case 'jan':
    console.log("Janeiro")
    break;
  case 'fev':
    console.log("Fevereiro")
    break;
}
```

```
    case 'mar':
        console.log("Macço")
        break;
    default:
        console.log("Mês não encontrado")
        break;
}
```

Quando for necessário que vários **cases** executem o mesmo bloco de código eles podem ser combinados:

```
let mesT = 3;

switch( mesT ) {
    case 1:
    case 2:
    case 3:
        console.log("1º Trimestre");
        break;
    default:
        console.log("Mês não encontrado");
        break;
}
```

Diferença entre switch e if/else

Número de Condições:

O **switch** é mais adequado quando há várias condições baseadas no valor de uma única expressão.

O **if/else** é mais flexível e pode lidar com uma variedade de condições diferentes, não limitado a uma única expressão.

Tipo de Comparação:

O **switch** faz comparação de igualdade estrita (===), o que significa que ele compara valores e tipos.

O **if/else** permite diferentes tipos de condições (>, <, >=, <=, ETC).

Comparação de Intervalos:

O **if/else** pode ser usado para comparar intervalos de valores ou condições mais complexas, enquanto o switch é mais adequado para comparações exatas.

Quando usar switch e quando usar if/else:

Em geral, a escolha entre **switch** e **if/else** depende da natureza específica do problema que você está resolvendo. Se a lógica condicional é simples e baseada em uma única expressão, **switch** pode ser mais limpo. Se a lógica é mais complexa, envolvendo várias expressões e condições, **if/else** pode ser mais apropriado.

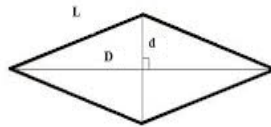
Exercícios

Exercício 82:

Crie uma aplicação para calcular e imprimir a área de um losango.

Se os valores de entrada forem negativos ou iguais a zero a aplicação deverá imprimir uma mensagem de erro - "Valores inválidos".

$$\text{Área} = (D * d) / 2$$



Exercício 83:

Conhecendo a diagonal maior e a diagonal menor de um losango regular, crie uma aplicação para calcular o valor de seu lado.

Atenção: Os valores não podem ser negativos ou iguais a zero.

Exercício 84:

Em época de pouco dinheiro, os comerciantes estão procurando aumentar suas vendas oferecendo desconto.

Faça uma aplicação que possa entrar com o valor de um produto e imprimir o novo valor tendo em vista que o desconto foi de 9%.

O valor do produto não pode ser menor ou igual a zero.

Exercício 85:

Construa uma aplicação que receba dois valores numéricos e efetue sua adição.

Caso o resultado da adição seja maior que 10, exibir os números digitados, o valor da adição e a raiz cúbica da adição.

Caso contrário exibir somente os valores digitados e o valor da adição.

Estrutura de repetição WHILE

A estrutura de repetição **while** é utilizada para executar um bloco de código repetidamente enquanto uma condição especificada for verdadeira.

A sintaxe do **while** é a seguinte:

```
while (condicao) {  
    // código a ser executado enquanto a condição for verdadeira  
}
```

O bloco de código dentro das chaves “{}” é repetidamente executado enquanto a condição fornecida dentro dos parênteses “()” for verdadeira.

A condição é avaliada antes da execução do bloco de código. Se a condição for falsa desde o início, o bloco de código nunca será executado.

Aqui temos um exemplo simples da estrutura **while** que imprime os números de 1 a 5:

```
let contador = 1;  
  
while (contador <= 5) {  
    console.log(contador);  
    contador++;  
}
```

Neste exemplo a variável **contador** é inicializada com 1.

A condição **contador <= 5** é avaliada antes de cada execução do bloco de código e enquanto a condição for verdadeira, o bloco de código é executado.

Dentro do bloco de código, o valor da variável **contador** é impresso e então a variável tem um incremento (**contador++**) para que a próxima iteração tenha um valor diferente.

Este loop continuará a ser executado até que a condição **contador <= 5** seja falsa. Uma vez que **contador** atinge o valor 6, a condição torna-se falsa, e o loop é encerrado.

É importante garantir que a condição eventualmente se torne falsa para evitar loops infinitos.

Exemplo de um loop infinito (evite fazer isso!)

```
while (true) {  
    console.log("Este é um loop infinito!");  
}
```

No exemplo acima, a condição é sempre *true*, então o loop continuaria indefinidamente, o que não é desejável na maioria dos casos. Portanto, certifique-se de definir condições que eventualmente se tornem falsas para garantir que o loop tenha uma conclusão.

Estrutura de repetição DO-WHILE

A estrutura de repetição **do-while** é uma forma de loop que executa um bloco de código enquanto uma condição especificada for verdadeira.

A diferença principal entre **do-while** e **while** é que o bloco de código dentro de **do-while** é executado pelo menos uma vez, mesmo que a condição seja falsa desde o início.

A sintaxe é a seguinte:

```
do {  
    // código a ser executado  
} while (condicao);
```

Exemplo:

```
let contador = 1;  
  
do {  
    console.log("Contagem: " + contador);  
    contador++;  
} while (contador <= 5);
```

Neste exemplo, o código dentro do bloco “do” será executado pelo menos uma vez, mesmo que a condição **contador <= 5** seja falsa desde o início.

Agora, vamos comparar **do-while** com a estrutura **while**:

Execução do Bloco:

Em **while**, o bloco de código só é executado se a condição for verdadeira desde o início.

Em **do-while**, o bloco de código é executado pelo menos uma vez, independentemente da condição ser verdadeira ou falsa inicialmente.

Avaliação da Condição:

Em **while**, a condição é avaliada antes da execução do bloco, o que significa que o bloco pode não ser executado se a condição for falsa desde o início.

Em **do-while**, a condição é avaliada após a execução do bloco, garantindo que o bloco seja executado pelo menos uma vez.

Quando usar **do-while** e quando usar **while**:

Use o **do-while** quando você deseja garantir que o bloco de código seja executado pelo menos uma vez, independentemente da condição inicial, neste caso a lógica dentro do bloco será executada antes de verificar a condição.

Use **while** quando você deseja executar o bloco de código apenas se a condição for verdadeira desde o início, neste caso a lógica dentro do bloco depende da condição ser verdadeira para ser executada.

Um cenário comum em que a estrutura de repetição **do-while** pode ser mais apropriada é quando você precisa solicitar a entrada de dados pelo menos uma vez e, em seguida, continuar a solicitar essa entrada até que ela atenda uma condição específica.

Vamos ver um exemplo em que estamos solicitando ao usuário para fornecer um número maior que 5:

```
let userInput;

do {

  userInput = prompt("Digite um número maior que 5: ");
  userInput = parseInt(userInput); // Converte a entrada para um número inteiro

} while (isNaN(userInput) || userInput <= 5);

console.log("Número válido: " + userInput);
```

Neste exemplo:

O bloco dentro de “**do**” solicita ao usuário para digitar um número.

A entrada do usuário é convertida para um número inteiro usando **parseInt**.

O loop **do-while** continuará a pedir que o usuário digite um novo número enquanto a entrada não for um número válido (verificado por **isNaN**) ou enquanto o número é menor ou igual a 5.

Este é um cenário onde a estrutura **do-while** é útil, pois garante que o bloco de código seja executado pelo menos uma vez, independentemente da validade da entrada inicial do usuário.

Se estivéssemos usando **while** puro, e a entrada inicial do usuário não fosse válida, o bloco de código não seria executado, o que pode não ser desejado em certas situações.

Break

O **Break** abandona o laço.

Observe:

```
let numeros = "
```

```
for (let i = 1; i <= 100; i++) {  
  numeros += i + ' - '  
  
  if (i % 11 == 0) {  
    break;  
  }  
}  
console.log(numeros)
```

Continue

O **continue** interrompe somente o laço atual permitindo o fluxo de execução da iteração continue.

Observe:

```
let numeros = ''  
  
for (let i = 1; i <= 100; i++) {  
  if (i % 2 == 0) {  
    continue  
  }  
  numeros += i + ' - '  
}  
  
console.log(numeros)
```

Exercícios

Exercício 86:

Crie uma aplicação para imprimir a soma de todos os números de 0 à 100.

Exercício 87:

Crie uma aplicação que receba um número obrigatoriamente maior que zero e imprima todos os números de zero até o número digitado.

Exercício 88:

Crie uma aplicação que receba um número obrigatoriamente menor que dez e imprima todos os números de vinte até o número digitado.

Exercício 89:

Crie uma aplicação que receba dois números e imprima a soma dos valores compreendidos entre estes números, inclusive os números digitados.

Exercício 90:

Crie uma aplicação que receba dois números e imprima a soma dos valores pares compreendidos entre estes números.

Exercício 91:

Crie uma aplicação que receba um número qualquer.

Se o número for positivo, imprimir o número digitado e sua raiz quadrada.

Se o número for negativo, imprimir o número digitado e seu valor elevado ao quadrado.

Exercício 92:

Crie uma aplicação que receba o nome e duas notas de um aluno.

As notas vão de zero a dez. O algoritmo deve imprimir o nome do aluno, suas notas e sua média.

Se a nota for maior que 7 – imprimir “Aprovado”.

Se a nota for menor que 5 – imprimir “Retido”.

Caso contrário - imprimir “Recuperação”.

Se as notas não estiverem no intervalo estabelecido o algoritmo deve emitir uma mensagem de erro.

Exercício 93

Uma sorveteria vende três tipos de picolés.

Sabendo-se que:

- ▶ O picolé do tipo 1 é vendido por R\$ 2.50
- ▶ O do tipo 2 por R\$ 5.60
- ▶ O do tipo 3 por R\$ 7.75.
- ▶ Crie uma aplicação que receba a quantidade comprada e o tipo do picolé (1, 2 ou 3) e imprima a quantidade vendida , o tipo, o preço e o total arrecadado. Considere a venda de apenas um tipo de picolé por vez.

Exercício 94

O cardápio de uma lanchonete é o seguinte:

<i>Código</i>	<i>Especificação</i>	<i>Preço unitário</i>
100	Cachorro quente	10,00
101	Bauru simples	11,50
102	Bauru c/ovo	15,50
103	Hambúrguer	16,50
104	Refrigerante	5,50

Crie uma aplicação que receba o item pedido, a quantidade e calcule o valor a ser pago por aquele lanche.

Considere que a cada execução somente será calculado um item.

Exercício 95

Criar uma aplicação que receba dois números diferentes e imprima os números compreendidos entre estes números, excluindo-se os números digitados.

Exercício 96

Calculando o IMC – Índice de Massa Corporal

Crie uma aplicação para calcular o índice de massa corporal basta utiliza a fórmula abaixo e comparar o resultado com a tabela abaixo.

$$\text{IMC} = \text{peso}/\text{altura}^2$$

Atenção: Peso em Kg e altura em metros

Tabela:

Abaixo de 17	Muito abaixo do peso
Entre 17 e 18,49	Abaixo do peso
Entre 18,5 e 24,99	Peso normal
Entre 25 e 29,99	Acima do peso
Entre 30 e 34,99	Obesidade I
Entre 35 e 39,99	Obesidade II (severa)
Acima de 40	Obesidade III (mórbida)

Observação:

O peso e a altura não podem ser nulos ou negativos.

Exercício 97

Um professor deseja criar uma aplicação pelo qual possa escolher que tipo de média deseja calcular a partir de **três notas**.

Você como programador deve criar esta aplicação que leia as notas, a opção escolhida pelo usuário e calcule a média:

- 1- aritmética
- 2- ponderada (pesos 3, 3, 4)

Exercício 98

Faça uma aplicação que calcule e mostre a soma dos 50 primeiros números pares positivos.

Exercício 99

Faça uma aplicação para imprimir todos os números pares, múltiplos de 5 e 7 ao mesmo tempo no intervalo de 1 até 1000. Imprima também a soma destes números.

Funções

De modo geral, **função** é um "subprograma" que pode ser chamado por código externo (ou interno no caso de recursão) à função.

Assim como o programa em si, uma função é composta por uma sequência de instruções chamada corpo da função.

Valores podem ser passados para uma função e ela vai retornar um valor.

Em JavaScript, funções são objetos de primeira classe, pois elas podem ter propriedades e métodos como qualquer outro objeto.

```
function nomeDaFuncao(){  
    //o que a função faz (corpo da função)  
}
```

Funções declaradas

O jeito mais básico de definir funções em JavaScript é através da função declarada (function declaration), toda função de declaração começa com a palavra reservada e obrigatória function, seguida pelo nome da função (também obrigatório) e uma lista de parâmetros (opcionais) separados por vírgula e encapsulados em parênteses (obrigatórios), o último passo é definir as chaves (obrigatórias) que será o corpo da função.

```
// criando a função  
function escrever(){  
    let idade = 18  
    let nome = 'fulano'  
    console.log(`Meu nome é ${nome} e eu tenho ${idade} de idade`);  
}
```



```
// Chamando a função
escrever()
```

Funções declaradas com parâmetros

```
// criando a função
function escrever(idade, nome){
  console.log(`Meu nome é ${nome} e eu tenho ${idade} de idade`);
}

let idade = 18
let nome = 'fulano'
// Chamando a função e passando parâmetros
escrever(idade, nome)
```

Expressões de função

A function expression (expressão de função) é muito parecida com a function declaration, a diferença é que uma função de expressão pode ser lidada como uma qualquer expressão em JavaScript, devendo ser atribuída a uma variável.

```
let nomeDaFuncao = function (){
  //o que a função faz (corpo da função)
}
```

Repare que é bem parecido com as funções de declaração, uma das súteis diferenças é que ela está sendo atribuída para uma variável, onde não definimos o nome da função e sim o nome da variável que irá referenciar a mesma.

```
// criando a função
let escrever = function(){
  let idade = 18
  let nome = 'fulano'
  console.log(`Meu nome é ${nome} e eu tenho ${idade} de idade`);
}

// Chamando a função
escrever()
```

Expressões de função com parâmetro

```
// criando a função
let escrever = function(nome, idade){
```

```
    console.log(`Meu nome é ${nome} e eu tenho ${idade} de idade`);
  }

  // Chamando a função
  let idade = 18
  let nome = 'fulano'
  escrever(idade, nome)
```

Arrow Functions

Arrow functions são simplificações para as functions expression..

```
let escrever = () => {
  let idade = 18
  let nome = 'fulano'
  console.log(`Meu nome é ${nome} e eu tenho ${idade} de idade`);
}

escrever()
```

Arrow Functions com parâmetros

```
let escrever = (idade, nome) => {
  console.log(`Meu nome é ${nome} e eu tenho ${idade} de idade`);
}

let idade = 18
let nome = 'fulano'
escrever(idade, nome)
```

Capturando dados de um formulário

HTML

Criando o formulário

```
<label for="nome">nome</label>
<input type="text" id="nome">

<label for="idade">idade</label>
<input type="text" id="idade">

<button id="btn">Escrever</button>
```

```
<div id="app"></div>
```

nome

idade

Escrever

JavaScript

Capturando os dados e exibindo os dados na tela.

```
// capturando os dados
let nome = document.getElementById('nome')
let idade = document.getElementById('idade')
let btn = document.getElementById('btn')
let div = document.getElementById('app')

// criando a função para exibir os dados
function escrever(){
  // exibindo os elementos no console
  console.log(nome);
  console.log(idade);
  console.log(btn);
  console.log(app);

  // inserindo os dados formatados no componente DIV
  // para pegar o valor devemos utilizar '.value'
  div.innerHTML = `O nome digitado foi ${nome.value} e a idade digitada foi
  ${idade.value}.`
}

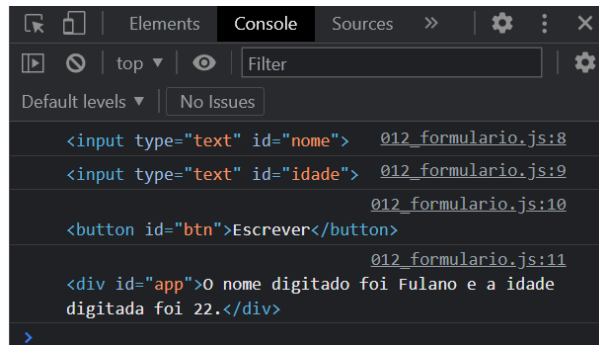
// vinculando o evento de clicar no botão para chamar a função escrever
btn.addEventListener('click', escrever)
```

Testando a aplicação

nome

idade

O nome digitado foi Fulano e a idade digitada foi 22.



```
<input type="text" id="nome">
<input type="text" id="idade">
<button id="btn">Escrever</button>
<div id="app">O nome digitado foi Fulano e a idade
digitada foi 22.</div>
```

Exercícios:

Exercício 100

Criar uma aplicação que leia um número que será o limite superior de um intervalo e imprimir todos os números ímpares menores do que esse número.

Exemplo:

Limite superior: 15

Saída: 1, 3, 5, 7, 9, 11, 13

Exercício 101

Criar uma aplicação que leia um número que servirá para controlar os números pares que serão impressos a partir de 2.

Exemplo:

Quantos: 4

Saída: 2, 4, 6, 8

Exercício 102

Criar uma aplicação que leia um número e imprima todos os números de 1 até o número lido e o seu produto.

Exemplo:

Número: 3

Saída: 1, 2, 3 - 6,

Exercício 103

Criar uma aplicação que leia um número (num) e imprima a soma dos números múltiplos de 5 no intervalo aberto entre 1 e num. Suponha que num será maior que zero.

Exemplo:

Limite superior: 15

Múltiplos de 5: 5 e 10

Saída: 15

Exercício 104

Criar uma aplicação que leia um número que servirá para controlar os primeiros números ímpares. Deverá ser impressa a soma desses números. Suponha que número será maior que zero.

Exemplo:

Quantos: 5

Primeiros ímpares: 1, 3, 5, 7, 9

Saída: 25

Exercício 105

Criar uma aplicação que leia os limites inferiores e superiores de um intervalo e imprima todos os números naturais no intervalo fechado em ordem crescente.

Exemplo:

Limite inferior: 5

Limite superior: 12

Saída: 5, 6, 7, 8, 9, 10, 11, 12

Exercício 106

Criar uma aplicação que leia os limites inferiores e superiores de um intervalo e imprimir todos os números múltiplos de 6 no intervalo fechado em ordem crescente.

Exemplo:

Limite inferior: 5

Limite superior: 13

Saída: 6, 12

Exercício 107

Criar uma aplicação que leia o limite inferior e superior de um intervalo e o número cujos múltiplos se deseja que sejam impressos no intervalo aberto (em ordem crescente).

Exemplo:

Limite inferior: 3

Limite superior: 12

Número: 3

Saída: 6, 9

Exercício 108

Criar uma aplicação que leia o limite inferior e superior de um intervalo e imprima todos os números pares no intervalo aberto e seu somatório.

Exemplo:

Limite inferior: 3

Limite superior: 12

Número: 3

Saída: 4, 6, 8, 10

Soma: 28

Exercício 109

Criar uma aplicação que leia um número (num) da entrada e imprima os múltiplos de 3 e 5 ao mesmo tempo no intervalo de 1 a num.

Exemplo.

Número lido: 50

Saída: 15, 30, 45

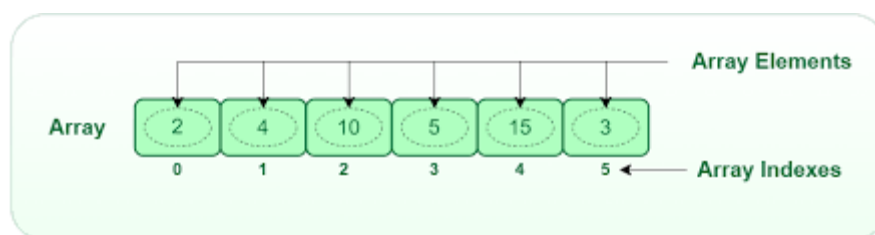
O que é um array?

Arrays são estruturas que servem para guardar dados, e organizá-los.

Seu objetivo é ser um espaço fixo na memória do computador para armazenar varios elementos.

Esses elementos podem ser acessados por um tipo de indicação, que chamamos de índice.

Arrays são variáveis compostas, unidimensionais e indexadas.



Declarando um array

```
let nomes = ['joao', 'jose', 'juca', 'anna']
```

Imprimindo seus dados

```
console.log(nomes[0]);  
console.log(nomes[1]);  
console.log(nomes[2]);
```

Exemplo: inserindo os dados em um array e exibindo seu valores no browser.

HTML

```
<label>Nome</label> <br>  
<input type="text" id="name"> <br><br>  
<button id="btn">cadastrar</button><br><br>  
<div id="app"></div>
```

JS

```

let nome = document.getElementById('name')
let app = document.getElementById('app')
let btn = document.getElementById('btn')

// Declarando um array
let nomes = []

function cadastrar(){

    // push - insere um valor no final do array
    nomes.push(nome.value)

    imprimir() // chamando a função imprimir
}

function imprimir(){
    console.log(nomes);
    app.innerHTML = "

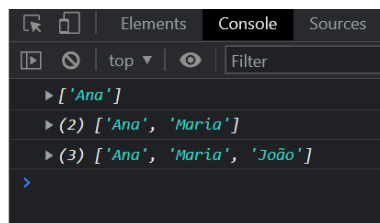
    // percorrendo um array
    for(let i = 0; i < nomes.length; i++){
        app.innerHTML += `Nome: ${nomes[i]} <br>`
    }
}

btn.addEventListener('click', cadastrar)

```

Nome

Nome: Ana
 Nome: Maria
 Nome: João



Abstração

Abstração é o processo de identificar as qualidades ou propriedades importantes do problema que está sendo modelado. Através de um modelo abstrato, pode-se concentrar nas características relevantes e ignorar as irrelevantes.

Objeto

Um objeto é um elemento computacional que representa alguma entidade (abstrata ou concreta) do problema sob análise. No paradigma de orientação a objetos, tudo pode ser potencialmente representado como um objeto.

Notação de um objeto literal em JavaScript

Um objeto literal é composto por um par de chaves " { } ", que envolve uma ou mais propriedades. Cada propriedade segue o formato " nome: valor " e devem ser separadas por vírgula.

No exemplo a seguir temos a abstração **aluno** escrito com a notação do JavaScript.

```
let aluno = {  
  nome: 'Cristiano Ronaldo',  
  idade: 22,  
  sexo: 'M'  
}  
  
console.log("Nome: " + aluno.nome);  
console.log("Idade: " + aluno.idade);  
console.log("Sexo: " + aluno.sexo);
```

Array de Objetos

HTML

```
<label>Nome:</label>  
<input type="text" id="nome" class="form-control">  
  
<label>Idade:</label>  
<input type="number" id="idade" class="form-control">  
  
<label>Sexo:</label>  
<select class="form-control" id="sexo">  
  <option value="">Selecione...</option>  
  <option value="Masculino">Masculino</option>  
  <option value="Feminino">Feminino</option>  
</select>  
  
<button id="btn" class="btn btn-primary">Cadastrar</button>  
  
<table class="table">  
  <thead>  
    <tr>  
      <th scope="col">NOME</th>  
      <th scope="col">IDADE</th>  
      <th scope="col">SEXO</th>  
    </tr>  
  </thead>  
  <tbody id="table">  
    <!-- Aqui será inserido os dados do array -->
```



```
        </tbody>
    </table>
</div>
</div>
```

JavaScript

```
let btn = document.getElementById('btn')
let tabela = document.getElementById('table')

// cria um array vazio
let arr = []

function cadastrar() {
    // pega os dados do formulário
    let nome = document.getElementById('nome').value
    let idade = document.getElementById('idade').value
    let sexo = document.getElementById('sexo').value

    // coloca os dados em um objeto literal
    let obj = {
        nome: nome,
        idade: idade,
        sexo: sexo
    }

    // armazena o objeto em um array
    arr.push(obj)

    // chama a função criarTabela
    criarTabela()
}

function criarTabela() {

    console.log(arr);

    tabela.innerHTML = ''

    for (let i = 0; i < arr.length; i++) {
        tabela.innerHTML += `
        <tr>
            <td>${arr[i].nome}</td>
            <td>${arr[i].idade}</td>
            <td>${arr[i].sexo}</td>
        </tr>
        `
    }
}
```

```

    }
    clear()
  }

function clear(){
  document.getElementById('nome').value = ""
  document.getElementById('idade').value = ""
  document.getElementById('sexo').value = ""
  document.getElementById('nome').focus()
}

btn.addEventListener('click', cadastrar)

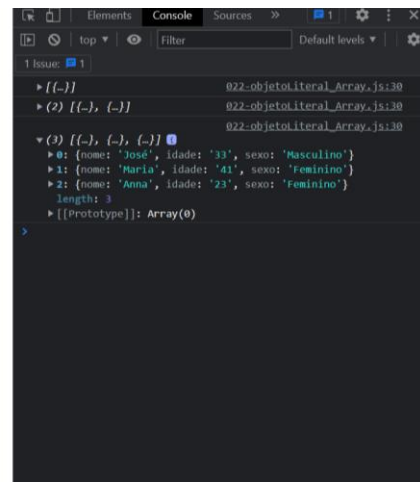
```

Array de Objeto

Nome: Idade:

Sexo:

NOME	IDADE	SEXO
José	33	Masculino
Maria	41	Feminino
Anna	23	Feminino



Exercícios

Exercício 110

Elabore uma aplicação que faça leitura de vários números inteiros e para cada vez que um número for inserido a aplicação retorne o maior e o menor número lido.

A aplicação também deve exibir todos os números inseridos.

Exercício 111

Faça uma aplicação que receba vários números, calcule e mostre:

- Todos os números digitados
- A soma dos números digitados
- A quantidade de números digitados
- A média dos números digitados
- A média dos números pares

Exercício 112

Faça uma aplicação que some todos os números abaixo de 1000 (até o número zero) que são múltiplos de 3 e de 5 ao mesmo tempo.

Exercício 113

Faça uma aplicação para imprimir em ordem decrescente todos os números múltiplos de 3 e de 5 compreendidos entre 30 e 300.

Exercício 114

Crie uma aplicação para ler dois números e imprimir todos os números pares e múltiplos de 7 (ao mesmo tempo) compreendidos entre os números digitados pelo usuário.

Atenção: o usuário pode digitar o primeiro número menor que o segundo e vice-versa.

Exercício 115

Criar uma aplicação que imprima a soma dos números ímpares de 1 a 200.

Exercício 116

Crie uma aplicação para imprimir todos os números ímpares de 1000 até 150 em ordem decrescente.

Exercício 117

Crie uma aplicação para imprimir uma tabela de conversão de polegadas para centímetros. Deseja-se que na tabela conste valores de 1 polegada até 20 polegadas inteiras.

Cada polegada possui 25,4 mm.

Exercício 118

Crie uma aplicação que receba uma palavra e imprima as letras da palavra separadas por um traço.

Exemplo:

Entrada: Guaratinguetá

Saída: G-u-a-r-a-t-i-n-g-u-e-t-á

Exercício 119

Criar uma aplicação que leia um número que será o limite superior de um intervalo e o incremento.

A aplicação deverá imprimir todos os números no intervalo de 0 até esse número.

Suponha que os dois números lidos são obrigatoriamente maiores que zero.

Exemplo:

- Limite superior: 20
- Incremento: 5
- Saída: 0, 5, 10, 15, 20

Exercício 120

Crie uma aplicação que, dado dois números informados pelo usuário (obrigatoriamente maior que zero e menor ou igual a cem), informe qual é o menor deles.

Exercício 121

Crie uma aplicação que, dado o nível de alerta de risco, imprima se ele for GRAVE. O nível de alerta é um número que varia de 0 a 10. O nível é considerado GRAVE quando ele é superior a 9.

Exercício 122

Crie uma aplicação para receber vários números (armazenados em um array) e imprima os números, os números pares e ímpares.

Exercício 123

Faça uma aplicação que leia uma variável e some 5 ao número caso ele seja par ou some 8 caso seja ímpar. Deve-se imprimir o resultado desta operação.

Exercício 124

Tendo como dados de entrada a altura e o sexo de uma pessoa, construa uma aplicação que calcule seu peso ideal, utilizando as seguintes fórmulas:

- para homens: $(72.7 * h) - 58$;
- para mulheres: $(62.1 * h) - 44.7$.

Exercício 125

Faça uma aplicação que percorra todos os números de 1 até 100. Para os números ímpares, deve ser impresso um "*", e para os números pares, deve ser impresso dois "@".

Exercício 126

Faça uma aplicação que percorra todos os números de 1 até 100. Para os números múltiplos de 4, imprima a palavra "PI" e para os outros imprima o próprio número.

Exercício 127

Crie uma aplicação que receba vários números e para cada número digitado a aplicação deve imprimir o número e a sua metade.

Exercício 128

Crie uma aplicação que receba vários números e para cada número digitado a aplicação deve exibir o número e o seu quadrado.

Exercício 129

Crie uma aplicação que receba um número qualquer maior que 10 e imprima todos os números de um até o número digitado em ordem crescente.

Exercício 130

Crie uma aplicação que receba um número qualquer menor que 10 e maior que zero e imprima todos os números de 100 até o número digitado em ordem decrescente.

Exercício 131

Crie uma aplicação que receba dois números diferentes e maiores que zero. A aplicação deve retornar todos os números múltiplos de 5 compreendidos entre o intervalo digitado em ordem crescente. Se nenhum número múltiplo de 5 for encontrado uma mensagem deve ser impressa [Não existem números múltiplos de 5]. Os números digitados não devem ser contabilizados.

Exercício 132

Crie uma aplicação que receba dois números diferentes e maiores que zero. A aplicação deve retornar todos os números múltiplos de 3 e 7 (ao mesmo tempo) compreendidos entre o intervalo digitado em ordem decrescente. Se nenhum número múltiplo de 3 e 7 for encontrado uma mensagem deve ser impressa [Não existem números múltiplos de 3 e 7]. Os números digitados devem ser contabilizados.

Exercício 133

Utilizando uma estrutura de repetição crie uma aplicação que imprima o quadrado dos números de 1 até 20.

Exercício 134

Utilizando uma estrutura de repetição crie uma aplicação que imprima os números pares no intervalo de 1 a 600

Exercício 135

Utilizando uma estrutura de repetição crie um algoritmo que leia um número que será o limite superior de um intervalo e imprimir todos os números ímpares menores do que esse número. Exemplo:

Limite superior: 15

Saída: 1 3 5 7 9 11 13

Exercício 136

Crie uma aplicação que leia um número que servirá para controlar os números pares que serão impressos a partir de 2.

Exemplo:

Quantos: 4

Saída: 2 4 6 8

Exercício 137

Crie uma aplicação que leia um número e imprima todos os números de 1 até o número lido e o seu produto.

Exemplo:

número: 3

Saída: 1 2 3

6 (produto)

Exercício 138

Utilizando a estrutura for crie um algoritmo que imprima a soma dos números pares entre 25 e 200.

Exercício 139

Crie uma aplicação que receba uma palavra qualquer [considere que a palavra não possui caracteres especiais].

Obs.: O ID do input deve ser o seu nome. O código deve compilar para ser avaliado.

- (1.50 pt) Exiba a palavra e posteriormente exiba a palavra separada por um traço;
- (1.50 pt) Exiba a quantidade de caracteres que a palavra contém
- (1.50 pt) Exiba a quantidade de vogais
- (1.50 pt) Exiba a quantidade e consoantes
- (2.00 pt) Exiba a palavra substituindo as vogais por um @
- (2.00 pt) Exiba a palavra substituindo as consoantes por um #

Exemplo:

Simulado

Escreva uma palavra:

Palavra digitada: Guaratingueta

Palavra Separada: G-u-a-r-a-t-i-n-g-u-e-t-a-

Quantidade de caracteres: 13

Quantidade de vogais: 7

Quantidade de consoantes: 6

Substituindo as vogais: G@@@r@t@ng@@t@

Substituindo as consoantes: #ua#a#i##ue#a|

Respostas:

Resposta exercício 001 até 010:

https://github.com/dionisioR/algorithmo_javascript_exercicio_001_010

Resposta exercícios 32:

Erro de Sintaxe

Resposta exercício 64:

A resposta correta é a alternativa B.

Explicação:

O código contém um erro de sintaxe na linha `if (numero % 2 = 0)`. O operador de comparação de igualdade deve ser `===` em vez de `=`.

Portanto, a linha correta deve ser `if (numero % 2 === 0)`, para verificar se `numero` é divisível por 2 e, assim, determinar se é par ou ímpar. O código funcionará corretamente após a correção desse erro.

Resposta exercício 65:

A resposta correta é a alternativa A.

Explicação:

O código está correto e funcionará como esperado. Ele verifica se `numero` é maior que 0 e, se for, imprime "Positivo"; caso contrário, imprime "Negativo".

Resposta exercício 66:

A resposta correta é a alternativa A.

Explicação:

O código está correto e funcionará como esperado. Ele calcula o IMC com base no peso e na altura, e em seguida, verifica se o IMC está dentro da faixa considerada saudável (entre 18.5 e 24.9, inclusos) e imprime "IMC saudável" ou "IMC não saudável" com base nessa verificação.

Resposta exercício 67:

A resposta correta é a alternativa D.

Explicação:

O código começa com `numCliques` inicializado como 0. Em seguida, ele incrementa `numCliques` usando o operador `++` (que incrementa o valor em 1) e depois adiciona 2 ao valor de `numCliques` usando `+= 2`. Portanto, após essas operações, `numCliques` será igual a 3. O valor impresso no console será "Número de cliques: 3".

Resposta exercício 68:

A resposta correta é a alternativa D.

Explicação:

O código começa com `itensVendidos` inicializado como 0. Em seguida, ele incrementa `itensVendidos` adicionando 3 e depois mais 2. Portanto, após essas operações, `itensVendidos` será igual a 5. O valor impresso no console será "Itens vendidos: 5".

Resposta exercício 69:

A resposta correta é a alternativa C.

Explicação:

O código começa com `estoqueProdutoA` inicializado como 10. Em seguida, ele decrementa `estoqueProdutoA` subtraindo 3 unidades e depois mais 2 unidades. Portanto, após essas operações, `estoqueProdutoA` será igual a 5. O valor impresso no console será "Quantidade de Produto A em estoque: 5".

Resposta exercício 70:

A resposta correta é a alternativa C.

Explicação:

O código começa com `contagemPessoas` inicializado como 0. Em seguida, ele incrementa `contagemPessoas` adicionando 3 pessoas, depois decrementa 1 pessoa, adiciona mais 2 pessoas e, por fim, decrementa 1 pessoa. Portanto, após essas operações, `contagemPessoas` será igual a 3. O valor impresso no console será "Número de pessoas na sala: 3".

Links:

Curso JavaScript:

<https://alunos.b7web.com.br/curso/javascript/js-introducao-ao-javascript>
<https://www.cursoemvideo.com/curso/javascript/>

Curso HTML e CSS:

<https://alunos.b7web.com.br/curso/html5-e-css3/o-que-e-html-e-pra-que-serve>
<https://www.cursoemvideo.com/curso/html5-css3-modulo1/>
<https://www.cursoemvideo.com/curso/curso-html5-e-css3-modulo-2-de-5-40-horas/>
<https://www.cursoemvideo.com/curso/curso-html5-e-css3-modulo-3-de-5-40-horas/>
<https://www.cursoemvideo.com/curso/curso-html5-e-css3-modulo-4-de-5-40-horas/>

Curso github:

<https://www.cursoemvideo.com/curso/curso-de-git-e-github/>

GITHUB:

<https://github.com/>

W3SCHOOLS:

<https://www.w3schools.com/>
<https://www.w3schools.com/js/default.asp>

DEVELOPER MOZILLA – JAVASCRIPT

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>

Plataforma para Treinar:

<https://www.beecrowd.com.br/judge/en/login>