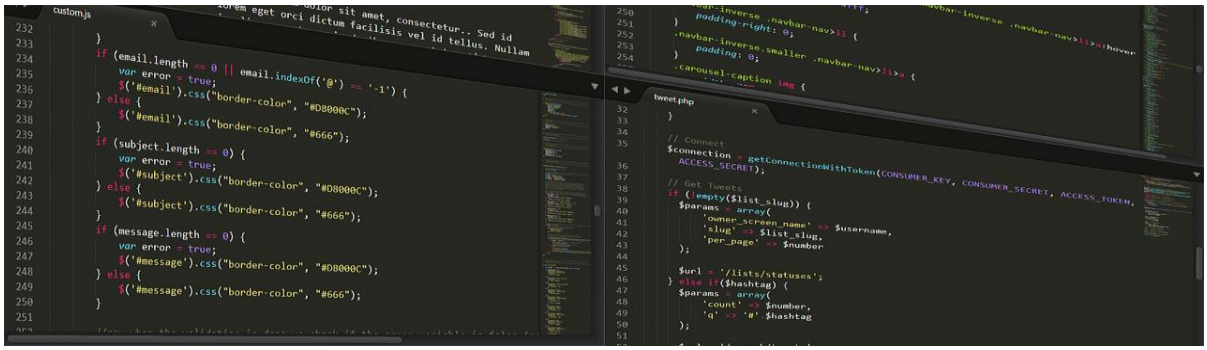
FUNDAÇÃO FAT

Lógica de Programação com C# e Java

Desenvolvedor de *Salesforce*



Sumário

Apresentação

Sejam bem-vindos a aula de nivelamento de conceitos de Lógica de Programação aplicada as Linguagens de Programação C# e Java, preparatório para o *Salesforce Platform Developer Credential*. Neste curso, iremos explorar os principais conceitos da Lógica de Programação utilizando como base um comparativo entre as linguagens C# e Java.

Os principais conceitos a serem abordados são:

- Introdução a lógica de Programação;
- Variáveis, constante e tipos de dados;
- Entradas e saídas de dados;
- Operadores matemáticos, relacionais e lógicos;
- Estruturas condicional e de seleção;
- Operador ternário;
- Estruturas de Repetição;
- Vetores e Listas;
- Funções e Procedimentos.

1. Introdução à Lógica de Programação

1.1. O que é Lógica de Programação?

Lógica de programação é a técnica de planejar a sequência de instruções que um computador deve seguir para resolver um problema ou executar uma tarefa específica. É a base para a criação de algoritmos, que são conjuntos de passos ordenados que descrevem como realizar uma tarefa.

Um algoritmo é uma sequência de instruções que utilizamos para solucionar um ou vários problemas, ou até mesmo realizar tarefas do dia a dia.

Um algoritmo não é necessariamente um programa computacional, pode ser passos que iremos tomar para realizar determinada tarefa.

O algoritmo deve sempre chegar ao resultado final esperado, caso não chegue, o mesmo não pode ser considerado finalizado.

1.2. A Importância da Lógica de Programação

Entender a lógica de programação é essencial para desenvolver programas eficientes e eficazes. A lógica bem estruturada permite resolver problemas de forma clara e ordenada, tornando mais fácil a leitura, manutenção e atualização do código.

1.3. Desenvolvimento de Programas Passo a Passo

Desenvolver um programa envolve a criação de um algoritmo que define passo a passo as ações que o computador deve realizar. Esses passos são escritos em uma linguagem de programação, mas antes de chegarmos a isso, é crucial pensar na lógica por trás das ações.

Vamos considerar exemplos do mundo real para entender melhor o conceito de passos e instruções estruturadas.

Exemplo do Mundo Real: Fazendo um Café.

1. Preparar os Ingredientes:
    - Pegue o café em pó.
    - Pegue a água.
    - Pegue o filtro de café.

- Pegue a cafeteira.

2. Ferver a Água:
   - Coloque a água na chaleira.
   - Ligue a chaleira e espere a água ferver.
3. Preparar o Filtro:
   - Coloque o filtro na cafeteira.
   - Adicione o café em pó no filtro.
4. Adicionar a Água:
   - Despeje a água fervida sobre o café no filtro.
5. Servir o Café:
   - Espere o café passar para a cafeteira.
   - Sirva o café na xícara.

Exemplo do Mundo Real: Trocando uma Lâmpada

1. Preparar o Ambiente:
   - Desligue o interruptor de luz.
   - Pegue uma nova lâmpada.
2. Remover a Lâmpada Queimada:
   - Suba em um banquinho, se necessário.
   - Gire a lâmpada queimada no sentido anti-horário até soltá-la do soquete.
3. Instalar a Nova Lâmpada:
   - Coloque a nova lâmpada no soquete.
   - Gire a nova lâmpada no sentido horário até fixá-la firmemente.
4. Testar a Nova Lâmpada:
   - Desça do banquinho, se necessário.
   - Ligue o interruptor de luz para verificar se a nova lâmpada está funcionando.

1.4. Aplicando Conceitos de Lógica de Programação

A lógica de programação segue princípios semelhantes aos exemplos do mundo real mencionados acima. Cada tarefa é decomposta em passos específicos que precisam ser seguidos em uma ordem lógica.

1.4.1. Sequência:

Cada instrução deve ser executada em uma ordem específica para alcançar o resultado desejado.

1.4.2. Decisão (Condicionais):

Em muitos casos, decisões precisam ser tomadas com base em condições. **Exemplo**: "Se a água estiver fervendo, então despeje-a no filtro de café."

1.4.3. Repetição (Loops):

Algumas tarefas precisam ser repetidas várias vezes. **Exemplo**: "Despeje a água sobre o café até que toda a água tenha passado pelo filtro."

Este capítulo forneceu uma introdução básica aos conceitos de lógica de programação, enfatizando a importância de pensar em termos de passos ordenados e estruturados.

2. Variáveis, Constantes e Tipos de Dados

Variável é um recurso utilizado nos programas computacionais para armazenar e recuperar dados, ou seja, é simplesmente um espaço na memória que reservamos atribuindo um nome, para que possamos organizar os dados à serem manipulados pelo programa. Por exemplo, podemos criar uma variável chamada “idade” para armazenar a idade de uma pessoa. Seria como você ter uma gaveta de escritório com repartições, onde a gaveta em si, seria a memória e cada repartição uma variável para armazenar algum objeto.

Uma **Constante** é responsável por armazenar um valor fixo em um espaço da memória, sendo que este valor que não se altera durante a execução do programa. Um exemplo clássico para uso de constante é em relação ao valor do PI. O valor fixo aproximado de PI é 3.14159265359. Sendo assim, em vez de sempre utilizarmos nas fórmulas o valor de PI, podemos definir uma constante que represente este valor durante a codificação.

Desta forma, podemos dizer que: “***Variáveis e constantes são como caixas onde você guarda valores. Variáveis são caixas que você pode abrir e trocar o que tem dentro, enquanto constantes são caixas lacradas que não podem ser alteradas depois de fechadas***.”

Exemplo C#:

```
int idade = 25; // Variável que pode ser alterada
const double PI = 3.14; // Constante que não pode ser alterada
```

Exemplo Java:

```
int idade = 25; // Variável que pode ser alterada
final double PI = 3.14; // Constante que não pode ser alterada
```

Os **Tipos de Dados** são a base para armazenar e manipular informações em qualquer linguagem de programação. Eles definem o tipo de valor que uma variável ou constante pode armazenar, como números inteiros, caracteres ou objetos. Vamos

comparar os tipos de dados disponíveis em C# e Java, destacando suas características e usos.

2.1. Tipos de Dados Primitivos

Os tipos de dados primitivos são os tipos básicos fornecidos pela linguagem. Eles não são objetos e armazenam valores simples.

Tipos Inteiros são tipos de dados que armazenam apenas valores inteiros, que podem ser positivos ou negativos. Possuem valores que variam de acordo com o tipo e a linguagem.

Tipo	C#	Java
byte	*byte*	*byte*
short	*short*	*short*
inteiro	*int*	*int*
longo	*long*	*long*
sem sinal*	*sbyte, ushort, uint, ulong*	**Não disponível nativamente**

* tipos sem sinal, ignoram a parte negativa que podem ser armazenada dobrando o valor positivo, ou seja, começam a contar em 0.

Exemplo C#:

```
byte b = 100;
short s = 10000;
int i = 100000;
long l = 10000000000L;
sbyte sb = -100;
ushort us = 20000;
uint ui = 4000000000U;
ulong ul = 10000000000000000000UL;
```

Exemplo Java:

```
byte b = 100;
short s = 10000;
int i = 100000;
long l = 10000000000L;
```

Tipos de Ponto Flutuante são usados para representar números reais (ou seja, números com partes fracionárias) e são particularmente eficientes em termos de desempenho e uso de memória. No entanto, eles são menos precisos do que os tipos decimais e podem introduzir pequenos erros de arredondamento, o que é aceitável para muitas aplicações, mas problemático para cálculos que exigem alta precisão.

Tipo	C#	Java
float	*float*	*float*
double	*double*	*double*

Exemplo C#:

```
float f = 3.14F;
double d = 3.141592653589793;
```

Exemplo Java:

```
float f = 3.14F;
double d = 3.141592653589793;
```

Tipo Decimal é projetado para cálculos financeiros e outras situações onde a precisão é crucial. Ele usa uma representação decimal exata, o que elimina os pequenos erros de arredondamento comuns com os tipos de ponto flutuante.

Tipo	C#	Java
decimal	*decimal*	*Não disponível diretamente (usar **BigDecimal**)*

Exemplo C#:

```
decimal dec = 100.25M;
```

Exemplo Java:

```
import java.math.BigDecimal;

BigDecimal dec = new BigDecimal("100.25");
```

Tipo Caractere como o próprio nome sugere, epresenta um único caractere lido, seja uma letra, número ou simbolo, mas do tipo textual.

Tipo	C#	Java
caractere	*char*	*char*

Exemplo C#:

```
char c = 'A';
```

Exemplo Java:

```
char c = 'A';
```

Tipo Booleano são tipos que permitem apenas dois valores, **verdadeiro** ou **falso**. Ele é usado principalmente para controlar o fluxo dos programas através de condições e expressões lógicas.

Tipo	C#	Java
booleano	*bool*	*boolean*

Exemplo C#:

```
bool isTrue = true;
```

Exemplo Java:

```
boolean isTrue = true;
```

2.2. Tipos de Dados Não Primitivos (Referenciais)

Os tipos de dados não primitivos são baseados em objetos e podem armazenar referências a dados e métodos.

Strings representam cadeias de caracteres e são utilizados para armazenas toda e qualquer informação textual, como nome, endereço, entre outras.

Tipo	C#	Java
string	*string*	*String*

Exemplo C#:

```
string mensagem = "Olá, mundo!";
```

Exemplo Java:

```
String mensagem = "Olá, mundo!";
```

Arrays em ambas as linguagens são usados para armazenar uma coleção de elementos do mesmo tipo.

Exemplo C#:

```
int[] numeros = {1, 2, 3, 4, 5};
string[] nomes = new string[3];
nomes[0] = "Alice";
nomes[1] = "Bob";
nomes[2] = "Carol";
```

Exemplo Java:

```
int[] numeros = {1, 2, 3, 4, 5};
String[] nomes = new String[3];
nomes[0] = "Alice";
nomes[1] = "Bob";
nomes[2] = "Carol";
```

Classes e Objetos em ambas as linguagens, você pode definir classes e criar objetos a partir delas. Uma **classe** é como um molde ou uma planta que define as características e comportamentos de algo. Pense em uma classe como um modelo que descreve um objeto, especificando quais propriedades (dados) ele tem e quais ações (métodos) ele pode realizar. Um **objeto** é uma instância de uma classe. Quando criamos um objeto, estamos criando algo baseado no modelo definido pela classe. Cada objeto pode ter seus próprios valores para as propriedades definidas na classe.

Exemplo C#:

```
public class Pessoa
{
    public string Nome { get; set; }
    public int Idade { get; set; }

    public void Saudacao()
    {
        Console.WriteLine($"Olá, meu nome é {Nome} e tenho {Idade} anos.");
    }
}

class Program
{
    static void Main()
    {
        Pessoa p = new Pessoa();
        p.Nome = "Alice";
        p.Idade = 30;
        p.Saudacao(); // Saída: Olá, meu nome é Alice e tenho 30 anos.
    }
}
```

Exemplo Java:

```java
public class Pessoa {
    private String nome;
    private int idade;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public int getIdade() {
        return idade;
    }

    public void setIdade(int idade) {
        this.idade = idade;
    }

    public void saudacao() {
        System.out.println("Olá, meu nome é " + nome + " e tenho " + idade + " anos.");
    }

    public static void main(String[] args) {
        Pessoa p = new Pessoa();
        p.setNome("Alice");
        p.setIdade(30);
        p.saudacao(); // Saída: Olá, meu nome é Alice e tenho 30 anos.
    }
}
```

Tanto **C#** quanto **Java** oferecem uma variedade de tipos de dados primitivos e não primitivos que são essenciais para armazenar e manipular informações. Embora existam muitas semelhanças entre as duas linguagens, algumas diferenças sutis, como a presença de tipos sem sinal em **C#** e o uso de **BigDecimal** para precisão decimal em **Java**, podem influenciar na escolha da linguagem para certos tipos de aplicações. Compreender essas diferenças e similaridades ajudará você a escrever código eficiente e a escolher a melhor linguagem para suas necessidades específicas.

3. Entrada e Saída de Dados

Em programação, a entrada de dados é o processo de receber informações do usuário, enquanto a saída de dados é o processo de exibir informações para o usuário. Imagine um caixa eletrônico: você insere seu cartão (entrada) e o caixa mostra seu saldo (saída). Em um programa, a entrada e saída de dados são essenciais para a interação com o usuário.

3.1. Entrada de Dados

A entrada de dados permite que o programa receba informações do usuário enquanto ele está sendo executado. Vamos ver como isso funciona nas linguagens **C#** e **Java**.

3.1.1. Entrada de Dados em C#

No **C#**, a entrada de dados é frequentemente feita usando a classe **Console**. O método **Console.ReadLine()** é usado para ler uma linha de texto digitada pelo usuário.

Exemplo:

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine("Digite seu nome:");
        string nome = Console.ReadLine(); // Lê o nome do usuário

        Console.WriteLine("Digite sua idade:");
        int idade = int.Parse(Console.ReadLine()); // Lê a idade e converte para inteiro

        Console.WriteLine("Olá, " + nome + "! Você tem " + idade + " anos.");
    }
}
```

3.1.2. Entrada de Dados em Java

No **Java**, a entrada de dados é geralmente feita usando a classe **Scanner**. O **Scanner** é usado para ler diferentes tipos de dados como strings e inteiros.

Exemplo:

```java
import java.util.Scanner;

public class EntradaDeDados {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Digite seu nome:");
        String nome = scanner.nextLine(); // Lê o nome do usuário

        System.out.println("Digite sua idade:");
        int idade = scanner.nextInt(); // Lê a idade como um inteiro

        System.out.println("Olá, " + nome + "! Você tem " + idade + " anos.");

        scanner.close();
    }
}
```

3.2. Saída de Dados

A saída de dados permite que o programa mostre informações ao usuário. Em ambos os exemplos acima, utilizamos a saída de dados para exibir mensagens e resultados.

3.2.1. Saída de Dados em C#

No **C#**, a saída de dados é feita usando a classe **Console** e os métodos **Console.WriteLine()** e **Console.Write().**

Exemplo:

```
using System;

class Program
{
    static void Main()
    {
        string mensagem = "Olá, mundo!";
        Console.WriteLine(mensagem); // Exibe a mensagem com uma nova linha

        int numero = 42;
        Console.Write("O número é: ");
        Console.Write(numero); // Exibe o número na mesma linha
    }
}
```

Saída de Dados em Java

No Java, a saída de dados é feita usando **System.out.println()** e **System.out.print().**

Exemplo:

```
public class SaidaDeDados {
    public static void main(String[] args) {
        String mensagem = "Olá, mundo!";
        System.out.println(mensagem); // Exibe a mensagem com uma nova linha

        int numero = 42;
        System.out.print("O número é: ");
        System.out.print(numero); // Exibe o número na mesma linha
    }
}
```

A entrada e saída de dados são fundamentais para a interação entre o programa e o usuário. Através dessas operações, podemos criar programas dinâmicos e interativos. Praticar a leitura e exibição de diferentes tipos de dados ajuda a entender como os programas processam e apresentam informações.

Além de **strings** e **inteiros**, podemos trabalhar com outros tipos de dados como números decimais e caracteres.

Exemplo C#:

```csharp
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine("Digite um número decimal:");
        double numeroDecimal = double.Parse(Console.ReadLine()); // Lê um número decimal

        Console.WriteLine("Digite um caractere:");
        char caractere = Console.ReadLine()[0]; // Lê um caractere

        Console.WriteLine("Você digitou o número " + numeroDecimal + " e o caractere '" + caractere + "'.");
    }
}
```

Exemplo Java:

```java
import java.util.Scanner;

public class EntradaSaidaDados {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Digite um número decimal:");
        double numeroDecimal = scanner.nextDouble(); // Lê um número decimal

        System.out.println("Digite um caractere:");
        char caractere = scanner.next().charAt(0); // Lê um caractere

        System.out.println("Você digitou o número " + numeroDecimal + " e o caractere '" + caractere + "'.");

        scanner.close();
    }
}
```

Continue praticando esses conceitos com diferentes tipos de dados e experimentando novas formas de interação com o usuário. Isso solidificará seu entendimento e melhorará suas habilidades de programação. Se tiver mais perguntas ou precisar de mais exemplos, estou aqui para ajudar!

4. Operadores Matemáticos

Operadores matemáticos são símbolos usados para realizar operações aritméticas em variáveis e valores. Eles são fundamentais para manipulação de dados numéricos em qualquer linguagem de programação. Neste capítulo, vamos explorar os operadores matemáticos básicos e avançados, com exemplos contextualizados em **C#** e **Java**.

4.1. Operadores Matemáticos Básicos

Os operadores matemáticos básicos incluem adição, subtração, multiplicação, divisão e módulo. Eles são utilizados para realizar operações aritméticas simples.

Operador	Descrição	C#	Java
+	Adição	+	+
-	Subtração	-	-
*	Multiplicação	*	*
/	Divisão	/	/
%	Módulo (Resto)	%	%

Vamos considerar um exemplo onde calculamos a média de notas de um aluno.

Em C#:

```
using System;

class Program
{
    static void Main()
    {
        int nota1 = 85;
        int nota2 = 90;
        int nota3 = 78;

        int soma = nota1 + nota2 + nota3;
        double media = soma / 3.0;  // Divisão precisa de ponto flutuante

        Console.WriteLine("A soma das notas é: " + soma);
        Console.WriteLine("A média das notas é: " + media);
    }
}
```

Em Java:

```java
public class CalculadoraNotas {
    public static void main(String[] args) {
        int nota1 = 85;
        int nota2 = 90;
        int nota3 = 78;

        int soma = nota1 + nota2 + nota3;
        double media = soma / 3.0;  // Divisão precisa de ponto flutuante

        System.out.println("A soma das notas é: " + soma);
        System.out.println("A média das notas é: " + media);
    }
}
```

4.2. Operadores de Incremento e Decremento

Os operadores de incremento e decremento são usados para aumentar ou diminuir o valor de uma variável em 1.

Operador	Descrição	C#	Java
++	Incremento	++	++
--	Decremento	--	--

Vamos considerar um exemplo onde contamos o número de passos de um pedômetro.

Em C#:

```csharp
using System;

class Program
{
    static void Main()
    {
        int passos = 0;

        // Simulando a contagem de passos
        passos++;
        passos++;
        passos--;

        Console.WriteLine("O número de passos é: " + passos);  // Saída: 1
    }
}
```

Em Java:

```java
public class ContadorPassos {
    public static void main(String[] args) {
        int passos = 0;

        // Simulando a contagem de passos
        passos++;
        passos++;
        passos--;

        System.out.println("O número de passos é: " + passos);  // Saída: 1
    }
}
```

4.3. Operadores Matemáticos Avançados

Além dos operadores básicos, existem operadores mais avançados que são frequentemente usados para cálculos matemáticos mais complexos.

4.3.1. Exponenciação

Embora **C#** e **Java** não tenham um operador específico para exponenciação, podemos usar métodos das bibliotecas padrão.

Em C#:

```csharp
using System;

class Program
{
    static void Main()
    {
        double baseNum = 2;
        double expoente = 3;
        double resultado = Math.Pow(baseNum, expoente);

        Console.WriteLine(baseNum + " elevado a " + expoente + " é: " + resultado); // Saída: 8
    }
}
```

Em Java:

```java
public class Exponenciacao {
    public static void main(String[] args) {
        double baseNum = 2;
        double expoente = 3;
        double resultado = Math.pow(baseNum, expoente);

        System.out.println(baseNum + " elevado a " + expoente + " é: " + resultado); // Saída: 8
    }
}
```

4.3.2. Raiz Quadrada

A raiz quadrada também é realizada usando métodos das bibliotecas padrão.

Em C#:

```csharp
using System;

class Program
{
    static void Main()
    {
        double numero = 16;
        double raizQuadrada = Math.Sqrt(numero);

        Console.WriteLine("A raiz quadrada de " + numero + " é: " + raizQuadrada);  // Saída: 4
    }
}
```

Em Java:

```java
public class RaizQuadrada {
    public static void main(String[] args) {
        double numero = 16;
        double raizQuadrada = Math.sqrt(numero);

        System.out.println("A raiz quadrada de " + numero + " é: " + raizQuadrada); // Saída: 4
    }
}
```

4.3.3. Operador de Módulo

O operador de módulo é usado para obter o resto de uma divisão. Isso é útil em diversas situações, como verificar se um número é par ou ímpar.

Em C#:

```csharp
using System;

class Program
{
    static void Main()
    {
        int numero = 10;
        int resto = numero % 3;

        Console.WriteLine("O resto de " + numero + " dividido por 3 é: " + resto);  // Saída: 1
    }
}
```

Em Java:

```java
public class OperadorModulo {
    public static void main(String[] args) {
        int numero = 10;
        int resto = numero % 3;

        System.out.println("O resto de " + numero + " dividido por 3 é: " + resto); // Saída: 1
    }
}
```

Os operadores matemáticos são essenciais para realizar cálculos e manipular dados numéricos em programação. Desde operações básicas como adição e subtração até operações mais complexas como exponenciação e raiz quadrada, esses operadores permitem que você desenvolva soluções eficazes para uma ampla variedade de problemas. Praticar o uso desses operadores em diferentes contextos ajudará a solidificar sua compreensão e habilidade em programação.

5. Operadores Relacionais

Os **operadores relacionais** são utilizados em programação para comparar dois valores e determinar a relação entre eles. Eles retornam um valor **booleano** (**true** ou **false**) indicando se a comparação é **verdadeira** ou **falsa**. Vamos explorar os operadores relacionais mais comuns e entender como usá-los em **C#** e **Java**.

Os operadores relacionais básicos são usados para comparar se dois valores são iguais, diferentes, maiores, menores, maiores ou iguais, ou menores ou iguais.

Operador	Descrição	Exemplo	Significado
==	Igual a	a == b	Verifica se a é igual a b.
!=	Diferente de	a != b	Verifica se a é diferente de b.
>	Maior que	a > b	Verifica se a é maior que b.
<	Menor que	a < b	Verifica se a é menor que b.
>=	Maior ou igual a	a >= b	Verifica se a é maior ou igual a b.
<=	Menor ou igual a	a <= b	Verifica se a é menor ou igual a b.

Vamos considerar um exemplo onde comparamos a idade de duas pessoas.

Em C#:

```
using System;

class Program
{
    static void Main()
    {
        int idadePessoa1 = 25;
        int idadePessoa2 = 30;

        bool saoMesmaIdade = idadePessoa1 == idadePessoa2;
        bool pessoa1MaisVelha = idadePessoa1 > idadePessoa2;

        Console.WriteLine("As idades são iguais: " + saoMesmaIdade);  // Saída: false
        Console.WriteLine("Pessoa 1 é mais velha: " + pessoa1MaisVelha);  // Saída: false
    }
}
```

Em Java:

```java
public class ComparadorIdades {
    public static void main(String[] args) {
        int idadePessoa1 = 25;
        int idadePessoa2 = 30;

        boolean saoMesmaIdade = idadePessoa1 == idadePessoa2;
        boolean pessoa1MaisVelha = idadePessoa1 > idadePessoa2;

        System.out.println("As idades são iguais: " + saoMesmaIdade);  // Saída: false
        System.out.println("Pessoa 1 é mais velha: " + pessoa1MaisVelha);  // Saída: false
    }
}
```

Os operadores relacionais são cruciais para comparar valores em programação. Eles nos permitem criar condições e tomar decisões com base em diferentes situações. Ao entender como esses operadores funcionam e praticar seu uso em diferentes contextos, você será capaz de escrever código mais eficaz e lógico em suas aplicações.

6. Operadores Lógicos

Os **operadores lógicos** são utilizados para combinar expressões booleanas e formar condições mais complexas. Eles são fundamentais para controlar o fluxo de execução em um programa, permitindo tomar decisões com base em múltiplas condições. Vamos explorar os operadores **E**, **OU** e **NÃO**, e entender como usá-los em **C#** e **Java**.

6.1. Operador E (AND)

O operador **E (AND)** retorna **verdadeiro** (**true**) somente se ambas as expressões comparadas forem verdadeiras.

Operador	Descrição	Exemplo	Resultado
&&	E (AND)	a && b	**true** se a e b são verdadeiros, senão **false**

Vamos considerar um exemplo onde verificamos se uma pessoa é maior de idade e tem carteira de motorista.

Em C#:

```
using System;

class Program
{
    static void Main()
    {
        int idade = 20;
        bool possuiCNH = true;

        bool podeDirigir = idade >= 18 && possuiCNH;

        Console.WriteLine("Pode dirigir? " + podeDirigir);  // Saída: true
    }
}
```

Em Java:

```java
public class VerificadorIdadeCNH {
    public static void main(String[] args) {
        int idade = 20;
        boolean possuiCNH = true;

        boolean podeDirigir = idade >= 18 && possuiCNH;

        System.out.println("Pode dirigir? " + podeDirigir);  // Saída: true
    }
}
```

6.2. Operador OU (OR)

O operador **OU (OR)** retorna **verdadeiro** (**true**) se pelo menos uma das expressões comparadas for verdadeira.

Operador	Descrição	Exemplo	Resultado
||	OU (OR)	a || b	**true** se a ou b são verdadeiros, senão **false**

Vamos considerar um exemplo onde verificamos se uma pessoa tem mais de 18 anos ou possui autorização dos pais para dirigir.

Em C#:

```csharp
using System;

class Program
{
    static void Main()
    {
        int idade = 17;
        bool autorizacaoPais = true;

        bool podeDirigir = idade >= 18 || autorizacaoPais;

        Console.WriteLine("Pode dirigir? " + podeDirigir);  // Saída: true
    }
}
```

Em Java:

```java
public class VerificadorIdadeAutorizacao {
    public static void main(String[] args) {
        int idade = 17;
        boolean autorizacaoPais = true;

        boolean podeDirigir = idade >= 18 || autorizacaoPais;

        System.out.println("Pode dirigir? " + podeDirigir);  // Saída: true
    }
}
```

6.3. Operador NÃO (NOT)

O operador **NÃO** (**NOT**) inverte o valor de uma expressão booleana, ou seja, se a expressão é **verdadeira**, o operador **NÃO** a torna **falsa** e vice-versa.

Operador	Descrição	Exemplo	Resultado
!	NÃO (NOT)	!a	**true** se a é falso, e **false** se a é verdadeiro

Vamos considerar um exemplo onde verificamos se uma pessoa não possui carteira de motorista.

Em C#:

```csharp
using System;

class Program
{
    static void Main()
    {
        bool possuiCNH = false;

        bool naoPodeDirigir = !possuiCNH;

        Console.WriteLine("Não pode dirigir? " + naoPodeDirigir);  // Saída: true
    }
}
```

Em Java:

```java
public class VerificadorCNH {
    public static void main(String[] args) {
        boolean possuiCNH = false;

        boolean naoPodeDirigir = !possuiCNH;

        System.out.println("Não pode dirigir? " + naoPodeDirigir);  // Saída: true
    }
}
```

Os **operadores lógicos E (AND), OU (OR) e NÃO (NOT)** são ferramentas poderosas para construir condições complexas em programas. Ao entender como esses operadores funcionam e como combiná-los de maneira eficaz, você será capaz de controlar o fluxo do seu programa de maneira mais precisa e eficiente. Vamos revisar alguns pontos importantes e apresentar exemplos adicionais para consolidar o aprendizado.

Resumo dos Operadores Lógicos:

- **E (AND) &&:** Retorna true se ambas as condições forem verdadeiras.
- **OU (OR) ||:** Retorna true se pelo menos uma das condições for verdadeira.
- **NÃO (NOT) !:** Inverte o valor da condição (se for true, torna false e vice-versa).

7. Estrutura Condicional e de Seleção

Estrutura condicional e de seleção são fundamentais em programação, permitindo que o fluxo do programa seja controlado com base em condições específicas. As duas estruturas mais comuns são **if** e **switch**. Vamos explorar cada uma delas em detalhes, com exemplos contextualizados em **C#** e **Java**.

7.1. Estrutura Condicional – if

Esta estrutura de controle, podemos definir como sendo o primeiro divisor de águas no mundo da programação, pois literalmente no momento da programação, é utilizando esta estrutura que entra uma palavra mágica chamada “se”, que dependendo do resultado da operação que for realizada, executa algumas instruções, “senão” executa outras instruções. Em outras palavras, permite a escolha de um grupo de instruções para serem executadas de acordo com a aceitação ou não de certas condições. São testados parâmetros e, dependendo de seus valores, é percorrido um caminho ou outro.

A **estrutura if** avalia uma expressão booleana e executa um bloco de código se a expressão for verdadeira. Opcionalmente, podemos usar **else** para definir um bloco de código a ser executado se a expressão for falsa, e **else if** para avaliar múltiplas condições.

Sintaxe Básica

C#:

```
if (condicao)
{
    // código a ser executado se a condição for verdadeira
}
else if (outraCondicao)
{
    // código a ser executado se a outraCondicao for verdadeira
}
else
{
    // código a ser executado se nenhuma condição anterior for verdadeira
}
```

Java:

```java
if (condicao) {
    // código a ser executado se a condição for verdadeira
} else if (outraCondicao) {
    // código a ser executado se a outraCondicao for verdadeira
} else {
    // código a ser executado se nenhuma condição anterior for verdadeira
}
```

Vamos considerar um exemplo onde determinamos a categoria de idade de uma pessoa.

Em C#:

```csharp
using System;

class Program
{
    static void Main()
    {
        int idade = 25;

        if (idade < 12)
        {
            Console.WriteLine("Criança");
        }
        else if (idade < 18)
        {
            Console.WriteLine("Adolescente");
        }
        else if (idade < 60)
        {
            Console.WriteLine("Adulto");
        }
        else
        {
            Console.WriteLine("Idoso");
        }
    }
}
```

Em Java:

```java
public class CategoriaIdade {
    public static void main(String[] args) {
        int idade = 25;

        if (idade < 12) {
            System.out.println("Criança");
        } else if (idade < 18) {
            System.out.println("Adolescente");
        } else if (idade < 60) {
            System.out.println("Adulto");
        } else {
            System.out.println("Idoso");
        }
    }
}
```

7.2. Estrutura de Seleção – switch

É uma estrutura de seleção múltipla que desvia o fluxo para um determinado ponto, de acordo com o valor das constantes determinadas em sua estrutura. Esta estrutura é muito utilizada principalmente em programas onde existe um menu de opções, onde o usuário deve escolher apenas uma, para ser executada.

Sintaxe Básica

C#:

```csharp
switch (expressao)
{
    case valor1:
        // código a ser executado se expressao == valor1
        break;
    case valor2:
        // código a ser executado se expressao == valor2
        break;
    // outros casos...
    default:
        // código a ser executado se nenhum caso for correspondido
        break;
}
```

Em Java:

```
switch (expressao) {
    case valor1:
        // código a ser executado se expressao == valor1
        break;
    case valor2:
        // código a ser executado se expressao == valor2
        break;
    // outros casos...
    default:
        // código a ser executado se nenhum caso for correspondido
        break;
}
```

Vamos considerar um exemplo onde determinamos o dia da semana com base em um número.

C#:

```
using System;

class Program
{
    static void Main()
    {
        int dia = 3;

        switch (dia)
        {
            case 1:
                Console.WriteLine("Domingo");
                break;
            case 2:
                Console.WriteLine("Segunda-feira");
                break;
            case 3:
                Console.WriteLine("Terça-feira");
                break;
            case 4:
                Console.WriteLine("Quarta-feira");
                break;
            case 5:
                Console.WriteLine("Quinta-feira");
                break;
```

```
            case 6:
                Console.WriteLine("Sexta-feira");
                break;
            case 7:
                Console.WriteLine("Sábado");
                break;
            default:
                Console.WriteLine("Número inválido");
                break;
        }
    }
}
```

Em Java:

```
public class DiaDaSemana {
    public static void main(String[] args) {
        int dia = 3;

        switch (dia) {
            case 1:
                System.out.println("Domingo");
                break;
            case 2:
                System.out.println("Segunda-feira");
                break;
            case 3:
                System.out.println("Terça-feira");
                break;
            case 4:
                System.out.println("Quarta-feira");
                break;
            case 5:
                System.out.println("Quinta-feira");
                break;
            case 6:
                System.out.println("Sexta-feira");
                break;
            case 7:
                System.out.println("Sábado");
                break;
            default:
                System.out.println("Número inválido");
                break;
        }
    }
}
```

Comparação entre if e switch

- Use **if** quando você precisa avaliar expressões booleanas complexas ou múltiplas condições diferentes.
- Use **switch** quando você está comparando um único valor contra várias possibilidades, especialmente quando essas possibilidades são constantes.

As estruturas condicionais **if** e **switch** são ferramentas poderosas para controlar o fluxo de execução em programas. Entender quando e como usar cada uma delas é essencial para escrever código limpo, eficiente e fácil de entender. Praticar com exemplos e exercícios ajudará a solidificar sua compreensão e habilidade em usar essas estruturas de maneira eficaz.

8. Operador Ternário

O **operador ternário** é uma forma concisa de expressar uma condição e retornar um valor com base nessa condição. Ele é útil quando você precisa atribuir um valor a uma variável com base em uma expressão booleana, e pode substituir **estruturas if-else simples**. Em **C#** e **Java**, **a sintaxe do operador ternário é a mesma**.

Sintaxe em C# e Java:

```
condicao ? valor_se_verdadeiro : valor_se_falso;
```

- **condicao**: Uma expressão booleana que é avaliada.
- **valor_se_verdadeiro**: O valor retornado se a condição for verdadeira.
- **valor_se_falso**: O valor retornado se a condição for falsa.

Vamos considerar um exemplo onde determinamos se uma pessoa é maior de idade ou menor de idade com base em sua idade.

Em C#:

```
using System;

class Program
{
    static void Main()
    {
        int idade = 20;
        string status = (idade >= 18) ? "Maior de idade" : "Menor de idade";
        Console.WriteLine("Status: " + status);  // Saída: Maior de idade
    }
}
```

Em Java:

```
public class VerificadorIdade {
    public static void main(String[] args) {
        int idade = 20;
        String status = (idade >= 18) ? "Maior de idade" : "Menor de idade";
        System.out.println("Status: " + status);  // Saída: Maior de idade
    }
}
```

Quando Usar o Operador Ternário

O operador ternário é ideal para situações onde você tem uma condição simples e deseja retornar um valor com base nessa condição. Ele pode tornar o código mais curto e legível, mas deve ser usado com cautela para não comprometer a clareza do código, especialmente em condições complexas.

Exemplo: Verificar se um Número é Par ou Ímpar

C#, sem operador ternário:

```
using System;

class Program
{
    static void Main()
    {
        int numero = 7;
        string resultado;
        if (numero % 2 == 0)
        {
            resultado = "Par";
        }
        else
        {
            resultado = "Ímpar";
        }
        Console.WriteLine("O número é: " + resultado);  // Saída: Ímpar
    }
}
```

C#, com operador ternário:

```
using System;

class Program
{
    static void Main()
    {
        int numero = 7;
        string resultado = (numero % 2 == 0) ? "Par" : "Ímpar";
        Console.WriteLine("O número é: " + resultado);  // Saída: Ímpar
    }
}
```

Java, sem operador ternário:

```java
public class VerificadorParidade {
    public static void main(String[] args) {
        int numero = 7;
        String resultado;
        if (numero % 2 == 0) {
            resultado = "Par";
        } else {
            resultado = "Ímpar";
        }
        System.out.println("O número é: " + resultado);  // Saída: Ímpar
    }
}
```

Java, com operador ternário:

```java
public class VerificadorParidade {
    public static void main(String[] args) {
        int numero = 7;
        String resultado = (numero % 2 == 0) ? "Par" : "Ímpar";
        System.out.println("O número é: " + resultado);  // Saída: Ímpar
    }
}
```

O **operador ternário** é uma ferramenta útil para simplificar expressões condicionais simples, tornando o código mais conciso. No entanto, para condições mais complexas, o uso de **if-else** pode ser mais adequado para manter a clareza e a legibilidade do código. Entender quando e como usar o operador ternário ajudará a escrever código mais limpo e eficiente.

9. Estruturas de Repetição

Utilizamos as **estruturas de repetição** quando desejamos que um determinado conjunto de instruções, sejam executados por mais de uma vez, podendo ser um número definido ou indefinido de vezes. Um exemplo simples de aplicação seria ao invés de usar 15 vezes a linha de comando escreva (“Sou programador!”); para exibir na tela esta mensagem 15 vezes, podemos utilizar um laço de repetição e automatizar este processo usando apenas 3 linhas de código.

Estruturas de repetição, ou loops, permitem que um bloco de código seja executado múltiplas vezes, com base em uma condição. Existem três estruturas de repetição principais em **C#** e **Java**: **for**, **while**, e **do-while**. Vamos explorar cada uma delas, explicando como funcionam, quando usá-las, e suas vantagens e desvantagens.

9.1. Estrutura de Repetição for

O **loop for** é usado quando sabemos de antemão quantas vezes queremos executar um bloco de código. Ele é composto por três partes: **inicialização**, **condição** e **incremento**.

Sintaxe Básica

C#:

```
for (inicializacao; condicao; incremento)
{
    // código a ser executado
}
```

Java:

```
for (inicializacao; condicao; incremento) {
    // código a ser executado
}
```

Vamos considerar um exemplo onde queremos imprimir os números de 1 a 10.

Em C#:

```csharp
using System;

class Program
{
    static void Main()
    {
        for (int i = 1; i <= 10; i++)
        {
            Console.WriteLine(i);
        }
    }
}
```

Em Java:

```java
public class ExemploFor {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            System.out.println(i);
        }
    }
}
```

Quando Usar

Use o **loop for** quando você souber o número exato de iterações que deseja realizar. É ideal para iterar sobre **arrays** ou **listas** com um número fixo de elementos.

Vantagens

- Clareza e simplicidade para loops de contagem.
- Fácil de entender e usar para loops com um número conhecido de iterações.

Desvantagens

- Não é tão flexível quanto while ou do-while para condições mais complexas ou quando o número de iterações não é conhecido de antemão.

9.2. Estrutura de Repetição while

O **loop while** é usado quando não sabemos de antemão quantas vezes queremos executar um bloco de código. Ele continua a executar enquanto a condição especificada for verdadeira.

Sintaxe Básica

C#:

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

Java:

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

Vamos considerar um exemplo onde queremos imprimir números de 1 a 10 usando **while**.

Em C#:

```
using System;

class Program
{
    static void Main()
    {
        int i = 1;
        while (i <= 10)
        {
            Console.WriteLine(i);
            i++;
        }
    }
}
```

Em Java:

```java
public class ExemploWhile {
    public static void main(String[] args) {
        int i = 1;
        while (i <= 10) {
            System.out.println(i);
            i++;
        }
    }
}
```

Quando Usar

Use o **loop while** quando o número de iterações não é conhecido e depende de uma condição que pode mudar durante a execução do loop.

Vantagens

- Flexível para condições que mudam durante a execução do **loop**.
- Útil para ler dados de entrada até que uma condição de parada seja encontrada.

Desvantagens

- Pode levar a **loops infinitos** se a **condição** nunca for **falsa**.
- Menos intuitivo para loops de contagem simples comparado ao for.

9.3. Estrutura de Repetição do-while

O **loop do-while** é semelhante ao **while**, mas garante que o bloco de código seja executado pelo menos uma vez antes da condição ser testada.

Sintaxe Básica

C#

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

Java:

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

Vamos considerar um exemplo onde queremos imprimir números de 1 a 10.

Em C#:

```
using System;

class Program
{
    static void Main()
    {
        int i = 1;
        do
        {
            Console.WriteLine(i);
            i++;
        }
        while (i <= 10);
    }
}
```

Em Java:

```
public class ExemploDoWhile {
    public static void main(String[] args) {
        int i = 1;
        do {
            System.out.println(i);
            i++;
        } while (i <= 10);
    }
}
```

Quando Usar

Use o **loop do-while** quando você deseja que o bloco de código seja executado pelo menos uma vez, independentemente da condição.

Vantagens

- Garante a execução do bloco de código pelo menos uma vez.
- Útil para menus interativos onde você deseja que a escolha do usuário seja processada pelo menos uma vez.

Desvantagens

- Menos utilizado que for e while.
- Pode ser menos intuitivo para alguns programadores, já que a condição é verificada no final.

9.4. Comparação das Estruturas de Repetição

- **for**: Melhor para loops de contagem conhecidos. Simples e claro para iterações fixas.
- **while**: Melhor para loops onde a condição de parada pode mudar durante a execução. Flexível para condições dinâmicas.
- **do-while**: Garante a execução do bloco de código pelo menos uma vez. Útil para casos específicos onde a execução inicial é necessária.

As estruturas de repetição são fundamentais para a criação de **loops** em programação. Entender quando e como usar cada tipo de **loop** (**for, while, do-while**) permite escrever código mais eficiente e adequado às necessidades específicas do seu programa. Praticar com diferentes exemplos ajudará a solidificar sua compreensão dessas estruturas essenciais.

10. Vetores e Listas

Vetores e **listas** são estruturas de dados essenciais em programação, utilizadas para armazenar coleções de elementos. **Vetores** (ou **arrays**) são de tamanho fixo e armazenam elementos do mesmo tipo, enquanto **listas** são mais **flexíveis**, permitindo operações dinâmicas como adição e remoção de elementos. Vamos explorar cada uma dessas estruturas em **C#** e **Java**.

10.1. Vetores (Arrays)

Um **vetor** é uma coleção de elementos de tamanho fixo, onde cada elemento pode ser acessado por um índice. Em **C#** e **Java**, os vetores são declarados de maneira semelhante.

Declaração e Inicialização

C#:

```
int[] numeros = new int[5]; // Declara um vetor de inteiros com 5 elementos
numeros[0] = 10; // Atribui valor ao primeiro elemento
int[] valores = { 1, 2, 3, 4, 5 }; // Inicializa um vetor com valores
```

Java:

```
int[] numeros = new int[5]; // Declara um vetor de inteiros com 5 elementos
numeros[0] = 10; // Atribui valor ao primeiro elemento
int[] valores = { 1, 2, 3, 4, 5 }; // Inicializa um vetor com valores
```

Vamos considerar um exemplo onde armazenamos e imprimimos as notas de um aluno.

Em C#:

```csharp
using System;

class Program
{
    static void Main()
    {
        int[] notas = { 85, 90, 78, 92, 88 };
        for (int i = 0; i < notas.Length; i++)
        {
            Console.WriteLine("Nota " + (i + 1) + ": " + notas[i]);
        }
    }
}
```

Em Java:

```java
public class NotasAluno {
    public static void main(String[] args) {
        int[] notas = { 85, 90, 78, 92, 88 };
        for (int i = 0; i < notas.length; i++) {
            System.out.println("Nota " + (i + 1) + ": " + notas[i]);
        }
    }
}
```

Vantagens dos Vetores

- **Simples e Rápido:** Acesso direto aos elementos usando índices.
- **Uso de Memória:** Boa utilização da memória para coleções de tamanho fixo.

Desvantagens dos Vetores

- **Tamanho Fixo:** Não podem ser redimensionados após a criação.
- **Inserção e Remoção:** Operações de inserção e remoção de elementos podem ser custosas, pois exigem deslocamento de elementos.

10.2. Listas

Listas são **coleções dinâmicas** que permitem o redimensionamento durante a execução do programa. Em **C#** e **Java**, as listas são implementadas através das classes **List** e **ArrayList**, respectivamente.

Declaração e Inicialização

C#

```
using System.Collections.Generic;

List<int> numeros = new List<int>();
numeros.Add(10); // Adiciona um elemento à lista
List<int> valores = new List<int> { 1, 2, 3, 4, 5 }; // Inicializa uma lista com valores
```

Java:

```
import java.util.ArrayList;

ArrayList<Integer> numeros = new ArrayList<>();
numeros.add(10); // Adiciona um elemento à lista
ArrayList<Integer> valores = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5)); // Inicializa uma
```

Vamos considerar um exemplo onde armazenamos e imprimimos as notas de um aluno, permitindo adicionar novas notas.

Em C#:

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
        List<int> notas = new List<int> { 85, 90, 78, 92, 88 };
        notas.Add(95); // Adiciona uma nova nota

        for (int i = 0; i < notas.Count; i++)
        {
            Console.WriteLine("Nota " + (i + 1) + ": " + notas[i]);
        }
    }
}
```

Em Java:

```java
import java.util.ArrayList;
import java.util.Arrays;

public class NotasAluno {
    public static void main(String[] args) {
        ArrayList<Integer> notas = new ArrayList<>(Arrays.asList(85, 90, 78, 92, 88));
        notas.add(95); // Adiciona uma nova nota

        for (int i = 0; i < notas.size(); i++) {
            System.out.println("Nota " + (i + 1) + ": " + notas.get(i));
        }
    }
}
```

Vantagens das Listas

- **Redimensionamento Dinâmico:** Podem crescer e encolher conforme necessário.
- **Facilidade de Uso:** Métodos convenientes para adição, remoção e busca de elementos.

Desvantagens das Listas

- **Desempenho:** Pode ser menos eficiente em termos de uso de memória e tempo de execução comparado a vetores para certos cenários.
- **Sobrecarga de Memória:** Podem consumir mais memória devido à capacidade adicional reservada para crescimento.

10.3. Comparação entre Vetores e Listas

- **Vetores**: Melhor para coleções de tamanho fixo e acesso rápido por índice.
- **Listas**: Melhor para coleções dinâmicas onde o tamanho pode mudar durante a execução.

Vetores e listas são fundamentais para o armazenamento e manipulação de coleções de dados em programação. Compreender suas características, vantagens e desvantagens permite escolher a estrutura de dados mais adequada para cada situação, resultando em programas mais eficientes e flexíveis. Praticar com exemplos e exercícios ajudará a solidificar sua compreensão dessas estruturas essenciais.

11. Funções e Procedimentos

A **modularização** consiste num método utilizado para facilitar a construção de grandes programas, através de uma divisão em pequenas etapas, que são os módulos ou subprogramas. A primeira delas, por onde começa a execução do trabalho recebe o nome de programa principal, e as outras são os subprogramas propriamente ditos, que são executados sempre que ocorre uma chamada dos mesmos, o que é feito através da especificação de seus nomes.

Funções e **procedimentos** são blocos de código reutilizáveis que executam tarefas específicas. A principal diferença entre eles é que funções retornam um valor, enquanto procedimentos não. Em **C#** e **Java**, esses blocos de código são conhecidos como métodos. Vamos explorar como declarar, chamar, e utilizar funções e procedimentos em ambas as linguagens.

11.1. Funções

Uma **função** é um bloco de código que realiza uma tarefa e retorna um valor. Funções são úteis para encapsular lógica que precisa ser reutilizada em diferentes partes do programa.

Declaração e Chamada de Funções

C#:

```
using System;

class Program
{
    // Função que soma dois números e retorna o resultado
    static int Somar(int a, int b)
    {
        return a + b;
    }

    static void Main()
    {
        int resultado = Somar(5, 3);
        Console.WriteLine("Resultado da soma: " + resultado);
    }
}
```

Java:

```java
public class ExemploFuncao {
    // Função que soma dois números e retorna o resultado
    public static int somar(int a, int b) {
        return a + b;
    }

    public static void main(String[] args) {
        int resultado = somar(5, 3);
        System.out.println("Resultado da soma: " + resultado);
    }
}
```

Vantagens das Funções

- **Reutilização**: Permite reutilizar código em diferentes partes do programa.
- **Modularidade**: Divide o programa em partes menores e mais manejáveis.
- **Manutenção**: Facilita a manutenção e atualização do código.

Desvantagens das Funções

- **Overhead**: Chamar funções pode introduzir um pequeno overhead de tempo de execução.
- **Complexidade**: Pode adicionar complexidade se não forem bem organizadas.

11.2. Procedimentos

Um **procedimento**, ou **método void**, é um bloco de código que realiza uma tarefa, mas não retorna um valor. Eles são usados para executar ações ou operações sem a necessidade de retorno.

Declaração e Chamada de Procedimentos

C#:

```csharp
using System;

class Program
{
    // Procedimento que imprime uma mensagem
    static void ImprimirMensagem(string mensagem)
    {
        Console.WriteLine(mensagem);
    }

    static void Main()
    {
        ImprimirMensagem("Olá, mundo!");
    }
}
```

Java:

```java
public class ExemploProcedimento {
    // Procedimento que imprime uma mensagem
    public static void imprimirMensagem(String mensagem) {
        System.out.println(mensagem);
    }

    public static void main(String[] args) {
        imprimirMensagem("Olá, mundo!");
    }
}
```

Vantagens dos Procedimentos

- **Simplicidade:** Ideal para operações que não requerem um valor de retorno.
- **Clareza**: Melhoram a clareza ao separar a lógica que não retorna valores.

Desvantagens dos Procedimentos

- **Sem Retorno:** Não podem ser usados quando um valor de retorno é necessário.

- **Estrutura**: Podem levar a uma estrutura de código menos clara se não forem bem organizados.

Vamos criar um exemplo de uma calculadora simples que usa funções e procedimentos para realizar operações básicas.

Em C#:

```
using System;

class Program
{
    // Função que soma dois números
    static int Somar(int a, int b)
    {
        return a + b;
    }

    // Função que subtrai dois números
    static int Subtrair(int a, int b)
    {
        return a - b;
    }

    // Procedimento que exibe o resultado
    static void ExibirResultado(string operacao, int resultado)
    {
        Console.WriteLine("Resultado da " + operacao + ": " + resultado);
    }

    static void Main()
    {
        int num1 = 10;
        int num2 = 5;

        int soma = Somar(num1, num2);
        ExibirResultado("soma", soma);

        int subtracao = Subtrair(num1, num2);
        ExibirResultado("subtração", subtracao);
    }
}
```

Em Java:

```java
public class CalculadoraSimples {
    // Função que soma dois números
    public static int somar(int a, int b) {
        return a + b;
    }

    // Função que subtrai dois números
    public static int subtrair(int a, int b) {
        return a - b;
    }

    // Procedimento que exibe o resultado
    public static void exibirResultado(String operacao, int resultado) {
        System.out.println("Resultado da " + operacao + ": " + resultado);
    }

    public static void main(String[] args) {
        int num1 = 10;
        int num2 = 5;

        int soma = somar(num1, num2);
        exibirResultado("soma", soma);

        int subtracao = subtrair(num1, num2);
        exibirResultado("subtração", subtracao);
    }
}
```

11.3. Comparação entre Funções e Procedimentos

- **Funções**: Usadas para realizar operações e retornar um valor. Ideal para cálculos e operações que produzem um resultado.
- **Procedimentos**: Usados para realizar operações sem retornar um valor. Ideal para operações que apenas executam uma ação, como imprimir valores ou modificar estados.

Funções e **procedimentos** são componentes essenciais para a criação de programas modulares e reutilizáveis. Entender suas diferenças e saber quando usá-los permite escrever código mais eficiente e organizado. Praticar com exemplos e exercícios ajudará a solidificar sua compreensão dessas estruturas fundamentais.

Exercícios Práticos

Com base nos conceitos estudados, desenvolva os códigos necessários a resolução das solicitações abaixo, de acordo com o material estudado. Você pode usar a linguagem que preferir. A resolução de cada exercício encontra-se ao final deste material, em **C#** e **Java**.

Operadores Matemáticos

Exercício 1: Leia a base e a altura de um triângulo e calcule a sua área usando a fórmula: área = (base * altura) / 2.

Exercício 2: Leia o raio de uma esfera e calcule o seu volume usando a fórmula: volume = (4/3) * π * raio³.

Exercício 3: Leia a temperatura em Celsius e converta para Fahrenheit usando a fórmula: fahrenheit = (celsius * 9/5) + 32.

Exercício 4: Calcule o montante final (capital + juros) após um determinado número de anos, utilizando juros compostos. Dados: capital (C), taxa de juros (r) e tempo (t). A fórmula é: montante = C * (1 + r/100)^t.

Exercício 5: Leia o comprimento e a largura de um retângulo e calcule o seu perímetro e área.

Estruturas Condicionais

Exercício 1: Leia um número e verifique se ele é par ou ímpar.

Exercício 2: Leia um número e verifique se ele é positivo, negativo ou zero.

Exercício 3: Leia dois números e exiba o maior deles.

Exercício 4: Leia a nota de um aluno e exiba o conceito (A, B, C, D, E) de acordo com a tabela:

A	B	C	D	E
90-100	80-89	70-79	60-69	0-59

Exercício 5: Leia um ano e verifique se ele é bissexto.

Estruturas de Repetição

Exercício 1: Leia um número N e exiba a soma de todos os números de 1 a N.

Exercício 2: Leia um número N e exiba o fatorial de N.

Exercício 3: Leia um número N e exiba todos os números pares de 1 a N.

Exercício 4: Leia um número N e exiba a tabuada de 1 a 10 desse número.

Exercício 5: Leia um número N e exiba uma contagem regressiva de N até 0.

Vetores e Listas

Exercício 1: Leia um vetor de 5 elementos e exiba a soma de seus elementos.

Exercício 2: Leia um vetor de 5 elementos e exiba o maior e o menor elemento.

Exercício 3: Leia um vetor de 5 elementos e um número N. Verifique se N está presente no vetor.

Exercício 4: Leia um vetor de 5 elementos e ordene seus elementos em ordem crescente.

Exercício 5: Leia uma lista de 5 elementos e exiba a média dos elementos.

Funções e Procedimentos

Exercício 1: Crie uma função que recebe dois números como parâmetros e retorna a potência do primeiro número elevado ao segundo.

Exercício 2: Crie um procedimento que recebe uma mensagem como parâmetro e a exibe na tela.

Exercício 3: Crie uma função que recebe um número como parâmetro e retorna true se ele for primo e false caso contrário.

Exercício 4: Crie um procedimento que recebe um número como parâmetro e imprime um triângulo com esse número de linhas.

Exercício 5: Crie uma função que recebe um número como parâmetro e retorna o fatorial desse número.

Operadores Matemáticos

Exercício 1: Calculando a Área de um Triângulo

C#:

```
using System;

class Program
{
    static void Main()
    {
        Console.Write("Digite a base do triângulo: ");
        double baseTriangulo = double.Parse(Console.ReadLine());

        Console.Write("Digite a altura do triângulo: ");
        double alturaTriangulo = double.Parse(Console.ReadLine());

        double areaTriangulo = (baseTriangulo * alturaTriangulo) / 2;
        Console.WriteLine("A área do triângulo é: " + areaTriangulo);
    }
}
```

Java:

```
import java.util.Scanner;

public class AreaTriangulo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite a base do triângulo: ");
        double baseTriangulo = scanner.nextDouble();

        System.out.print("Digite a altura do triângulo: ");
        double alturaTriangulo = scanner.nextDouble();

        double areaTriangulo = (baseTriangulo * alturaTriangulo) / 2;
        System.out.println("A área do triângulo é: " + areaTriangulo);
    }
}
```

Operadores Matemáticos

Exercício 2: Volume de uma Esfera

C#:

```
using System;

class Program
{
    static void Main()
    {
        double pi = Math.PI;

        Console.Write("Digite o raio da esfera: ");
        double raio = double.Parse(Console.ReadLine());

        double volume = (4.0 / 3.0) * pi * Math.Pow(raio, 3);
        Console.WriteLine("O volume da esfera é: " + volume);
    }
}
```

Java:

```
import java.util.Scanner;

public class VolumeEsfera {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        double pi = Math.PI;

        System.out.print("Digite o raio da esfera: ");
        double raio = scanner.nextDouble();

        double volume = (4.0 / 3.0) * pi * Math.pow(raio, 3);
        System.out.println("O volume da esfera é: " + volume);
    }
}
```

Operadores Matemáticos

Exercício 3: Conversão de Temperatura

C#:

```
using System;

class Program
{
    static void Main()
    {
        Console.Write("Digite a temperatura em Celsius: ");
        double celsius = double.Parse(Console.ReadLine());

        double fahrenheit = (celsius * 9 / 5) + 32;
        Console.WriteLine("A temperatura em Fahrenheit é: " + fahrenheit);
    }
}
```

Java:

```
import java.util.Scanner;

public class ConversaoTemperatura {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite a temperatura em Celsius: ");
        double celsius = scanner.nextDouble();

        double fahrenheit = (celsius * 9 / 5) + 32;
        System.out.println("A temperatura em Fahrenheit é: " + fahrenheit);
    }
}
```

Operadores Matemáticos

Exercício 4: Juros Compostos

C#:

```
using System;

class Program
{
    static void Main()
    {
        Console.Write("Digite o capital inicial: ");
        double capital = double.Parse(Console.ReadLine());

        Console.Write("Digite a taxa de juros (em %): ");
        double taxaJuros = double.Parse(Console.ReadLine());

        Console.Write("Digite o número de anos: ");
        int anos = int.Parse(Console.ReadLine());

        double montante = capital * Math.Pow((1 + taxaJuros / 100), anos);
        Console.WriteLine("O montante final é: " + montante);
    }
}
```

Java:

```
import java.util.Scanner;

public class JurosCompostos {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite o capital inicial: ");
        double capital = scanner.nextDouble();

        System.out.print("Digite a taxa de juros (em %): ");
        double taxaJuros = scanner.nextDouble();

        System.out.print("Digite o número de anos: ");
        int anos = scanner.nextInt();

        double montante = capital * Math.pow((1 + taxaJuros / 100), anos);
        System.out.println("O montante final é: " + montante);
    }
}
```

Operadores Matemáticos

Exercício 5: Perímetro e Área de um Retângulo

C#:

```
using System;

class Program
{
    static void Main()
    {
        Console.Write("Digite o comprimento do retângulo: ");
        double comprimento = double.Parse(Console.ReadLine());

        Console.Write("Digite a largura do retângulo: ");
        double largura = double.Parse(Console.ReadLine());

        double perimetro = 2 * (comprimento + largura);
        double area = comprimento * largura;

        Console.WriteLine("O perímetro do retângulo é: " + perimetro);
        Console.WriteLine("A área do retângulo é: " + area);
    }
}
```

Java:

```
import java.util.Scanner;

public class PerimetroAreaRetangulo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite o comprimento do retângulo: ");
        double comprimento = scanner.nextDouble();

        System.out.print("Digite a largura do retângulo: ");
        double largura = scanner.nextDouble();

        double perimetro = 2 * (comprimento + largura);
        double area = comprimento * largura;

        System.out.println("O perímetro do retângulo é: " + perimetro);
        System.println("A área do retângulo é: " + area);
    }
}
```

Estruturas Condicionais

Exercício 1: Par ou Ímpar

C#:

```
using System;

class Program
{
    static void Main()
    {
        Console.Write("Digite um número: ");
        int numero = int.Parse(Console.ReadLine());

        if (numero % 2 == 0)
        {
            Console.WriteLine("O número é par.");
        }
        else
        {
            Console.WriteLine("O número é ímpar.");
        }
    }
}
```

Java:

```
import java.util.Scanner;

public class ParOuImpar {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um número: ");
        int numero = scanner.nextInt();

        if (numero % 2 == 0) {
            System.out.println("O número é par.");
        } else {
            System.out.println("O número é ímpar.");
        }
    }
}
```

Estruturas Condicionais

Exercício 2: Positivo, Negativo ou Zero

C#:

```
using System;

class Program
{
    static void Main()
    {
        Console.Write("Digite um número: ");
        int numero = int.Parse(Console.ReadLine());

        if (numero > 0)
        {
            Console.WriteLine("O número é positivo.");
        }
        else if (numero < 0)
        {
            Console.WriteLine("O número é negativo.");
        }
        else
        {
            Console.WriteLine("O número é zero.");
        }
    }
}
```

Java:

```
import java.util.Scanner;

public class PositivoNegativoZero {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um número: ");
        int numero = scanner.nextInt();

        if (numero > 0) {
            System.out.println("O número é positivo.");
        } else if (numero < 0) {
            System.out.println("O número é negativo.");
        } else {
            System.out.println("O número é zero.");
        }
    }
}
```

Estruturas Condicionais

Exercício 3: Maior de Dois Números

C#:

```
using System;

class Program
{
    static void Main()
    {
        Console.Write("Digite o primeiro número: ");
        int num1 = int.Parse(Console.ReadLine());

        Console.Write("Digite o segundo número: ");
        int num2 = int.Parse(Console.ReadLine());

        if (num1 > num2)
        {
            Console.WriteLine("O maior número é: " + num1);
        }
        else if (num2 > num1)
        {
            Console.WriteLine("O maior número é: " + num2);
        }
        else
        {
            Console.WriteLine("Os números são iguais.");
        }
    }
}
```

Java:

```
import java.util.Scanner;

public class MaiorDeDoisNumeros {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite o primeiro número: ");
        int num1 = scanner.nextInt();

        System.out.print("Digite o segundo número: ");
        int num2 = scanner.nextInt();

        if (num1 > num2) {
            System.out.println("O maior número é: " + num1);
        } else if (num2 > num1) {
            System.out.println("O maior número é: " + num2);
        } else {
            System.out.println("Os números são iguais.");
        }
    }
}
```

Estruturas Condicionais

Exercício 4: Notas e Conceito

C#:

```
using System;

class Program
{
    static void Main()
    {
        Console.Write("Digite a nota do aluno: ");
        int nota = int.Parse(Console.ReadLine());

        if (nota >= 90)
        {
            Console.WriteLine("Conceito: A");
        }
        else if (nota >= 80)
        {
            Console.WriteLine("Conceito: B");
        }
        else if (nota >= 70)
        {
            Console.WriteLine("Conceito: C");
        }
        else if (nota >= 60)
        {
            Console.WriteLine("Conceito: D");
        }
        else
        {
            Console.WriteLine("Conceito: E");
        }
    }
}
```

Java:

```
import java.util.Scanner;

public class NotasConceito {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite a nota do aluno: ");
        int nota = scanner.nextInt();

        if (nota >= 90) {
            System.out.println("Conceito: A");
        } else if (nota >= 80) {
            System.out.println("Conceito: B");
        } else if (nota >= 70) {
            System.out.println("Conceito: C");
        } else if (nota >= 60) {
            System.out.println("Conceito: D");
        } else {
            System.out.println("Conceito: E");
        }
    }
}
```

Estruturas Condicionais

Exercício 5: Verificação de Ano Bissexto

C#:

```
using System;

class Program
{
    static void Main()
    {
        Console.Write("Digite um ano: ");
        int ano = int.Parse(Console.ReadLine());

        if ((ano % 4 == 0 && ano % 100 != 0) || ano % 400 == 0)
        {
            Console.WriteLine("O ano é bissexto.");
        }
        else
        {
            Console.WriteLine("O ano não é bissexto.");
        }
    }
}
```

Java:

```
import java.util.Scanner;

public class VerificacaoAnoBissexto {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um ano: ");
        int ano = scanner.nextInt();

        if ((ano % 4 == 0 && ano % 100 != 0) || ano % 400 == 0) {
            System.out.println("O ano é bissexto.");
        } else {
            System.out.println("O ano não é bissexto.");
        }
    }
}
```

Estruturas de Repetição

Exercício 1: Soma de Números de 1 a N

C#:

```
using System;

class Program
{
    static void Main()
    {
        Console.Write("Digite um número: ");
        int n = int.Parse(Console.ReadLine());

        int soma = 0;
        for (int i = 1; i <= n; i++)
        {
            soma += i;
        }

        Console.WriteLine("A soma de 1 a " + n + " é: " + soma);
    }
}
```

Java:

```
import java.util.Scanner;

public class SomaDe1AN {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um número: ");
        int n = scanner.nextInt();

        int soma = 0;
        for (int i = 1; i <= n; i++) {
            soma += i;
        }

        System.out.println("A soma de 1 a " + n + " é: " + soma);
    }
}
```

Estruturas de Repetição

Exercício 2: Fatorial de um Número

C#:

```
using System;

class Program
{
    static void Main()
    {
        Console.Write("Digite um número: ");
        int n = int.Parse(Console.ReadLine());

        int fatorial = 1;
        for (int i = 1; i <= n; i++)
        {
            fatorial *= i;
        }

        Console.WriteLine("O fatorial de " + n + " é: " + fatorial);
    }
}
```

Java:

```
import java.util.Scanner;

public class Fatorial {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um número: ");
        int n = scanner.nextInt();

        int fatorial = 1;
        for (int i = 1; i <= n; i++) {
            fatorial *= i;
        }

        System.out.println("O fatorial de " + n + " é: " + fatorial);
    }
}
```

Estruturas de Repetição

Exercício 3: Números Pares de 1 a N

C#:

```
using System;

class Program
{
    static void Main()
    {
        Console.Write("Digite um número: ");
        int n = int.Parse(Console.ReadLine());

        for (int i = 1; i <= n; i++)
        {
            if (i % 2 == 0)
            {
                Console.WriteLine(i);
            }
        }
    }
}
```

Java:

```
import java.util.Scanner;

public class NumerosPares {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um número: ");
        int n = scanner.nextInt();

        for (int i = 1; i <= n; i++) {
            if (i % 2 == 0) {
                System.out.println(i);
            }
        }
    }
}
```

Estruturas de Repetição

Exercício 4: Tabuada de um Número

C#:

```
using System;

class Program
{
    static void Main()
    {
        Console.Write("Digite um número: ");
        int n = int.Parse(Console.ReadLine());

        for (int i = 1; i <= 10; i++)
        {
            Console.WriteLine(n + " x " + i + " = " + (n * i));
        }
    }
}
```

Java:

```
import java.util.Scanner;

public class Tabuada {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um número: ");
        int n = scanner.nextInt();

        for (int i = 1; i <= 10; i++) {
            System.out.println(n + " x " + i + " = " + (n * i));
        }
    }
}
```

Estruturas de Repetição

Exercício 5: Contagem Regressiva

C#:

```
using System;

class Program
{
    static void Main()
    {
        Console.Write("Digite um número: ");
        int n = int.Parse(Console.ReadLine());

        for (int i = n; i >= 0; i--)
        {
            Console.WriteLine(i);
        }
    }
}
```

Java:

```
import java.util.Scanner;

public class ContagemRegressiva {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um número: ");
        int n = scanner.nextInt();

        for (int i = n; i >= 0; i--) {
            System.out.println(i);
        }
    }
}
```

Vetores e Listas

Exercício 1: Soma de Elementos de um Vetor

C#:

```
using System;

class Program
{
    static void Main()
    {
        int[] vetor = new int[5] { 1, 2, 3, 4, 5 };
        int soma = 0;

        for (int i = 0; i < vetor.Length; i++)
        {
            soma += vetor[i];
        }

        Console.WriteLine("A soma dos elementos do vetor é: " + soma);
    }
}
```

Java:

```
import java.util.Scanner;

public class SomaElementosVetor {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int[] vetor = new int[5];
        int soma = 0;

        for (int i = 0; i < vetor.length; i++) {
            System.out.print("Digite o elemento " + (i + 1) + ": ");
            vetor[i] = scanner.nextInt();
            soma += vetor[i];
        }

        System.out.println("A soma dos elementos do vetor é: " + soma);
    }
}
```

Vetores e Listas

Exercício 2: Maior e Menor Elemento de um Vetor

C#:

```
using System;

class Program
{
    static void Main()
    {
        int[] vetor = new int[5] { 10, 5, 20, 15, 25 };
        int maior = vetor[0];
        int menor = vetor[0];

        for (int i = 1; i < vetor.Length; i++)
        {
            if (vetor[i] > maior)
            {
                maior = vetor[i];
            }

            if (vetor[i] < menor)
            {
                menor = vetor[i];
            }
        }

        Console.WriteLine("O maior elemento do vetor é: " + maior);
        Console.WriteLine("O menor elemento do vetor é: " + menor);
    }
}
```

Java:

```
import java.util.Scanner;

public class MaiorMenorElementoVetor {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int[] vetor = new int[5];
        int maior = vetor[0];
        int menor = vetor[0];

        for (int i = 0; i < vetor.length; i++) {
            System.out.print("Digite o elemento " + (i + 1) + ": ");
            vetor[i] = scanner.nextInt();

            if (vetor[i] > maior) {
                maior = vetor[i];
            }

            if (vetor[i] < menor) {
                menor = vetor[i];
            }
        }

        System.out.println("O maior elemento do vetor é: " + maior);
        System.out.println("O menor elemento do vetor é: " + menor);
    }
}
```

Vetores e Listas

Exercício 3: Busca de Elemento em um Vetor

C#:

```
using System;

class Program
{
    static void Main()
    {
        int[] vetor = new int[5] { 10, 20, 30, 40, 50 };
        int numero = 30;
        bool encontrado = false;

        for (int i = 0; i < vetor.Length; i++)
        {
            if (vetor[i] == numero)
            {
                encontrado = true;
                break;
            }
        }

        if (encontrado)
        {
            Console.WriteLine("O número " + numero + " está presente no vetor.");
        }
        else
        {
            Console.WriteLine("O número " + numero + " não está presente no vetor.");
        }
    }
}
```

Java:

```
import java.util.Scanner;

public class BuscaElementoVetor {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int[] vetor = new int[5];
        System.out.print("Digite 5 números separados por espaço: ");
        for (int i = 0; i < vetor.length; i++) {
            vetor[i] = scanner.nextInt();
        }

        System.out.print("Digite um número para buscar: ");
```

```
        int numero = scanner.nextInt();
        boolean encontrado = false;

        for (int i = 0; i < vetor.length; i++) {
            if (vetor[i] == numero) {
                encontrado = true;
                break;
            }
        }

        if (encontrado) {
            System.out.println("O número " + numero + " está presente no vetor.");
        } else {
            System.out.println("O número " + numero + " não está presente no
vetor.");
        }
    }
}
```

Vetores e Listas

Exercício 4: Ordenação de um Vetor

C#:

```
using System;

class Program
{
    static void Main()
    {
        int[] vetor = new int[5] { 40, 10, 30, 20, 50 };

        Array.Sort(vetor);

        Console.WriteLine("Vetor ordenado em ordem crescente:");
        foreach (int elemento in vetor)
        {
            Console.Write(elemento + " ");
        }
    }
}
```

Java:

```
import java.util.Arrays;
import java.util.Scanner;

public class OrdenacaoVetor {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int[] vetor = new int[5];
        System.out.print("Digite 5 números separados por espaço: ");
        for (int i = 0; i < vetor.length; i++) {
            vetor[i] = scanner.nextInt();
        }

        Arrays.sort(vetor);

        System.out.println("Vetor ordenado em ordem crescente:");
        for (int elemento : vetor) {
            System.out.print(elemento + " ");
        }
    }
}
```

Vetores e Listas

Exercício 5: Média de Elementos de uma Lista

C#:

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
        List<int> lista = new List<int>() { 10, 20, 30, 40, 50 };
        int soma = 0;

        foreach (int elemento in lista)
        {
            soma += elemento;
        }

        double media = (double)soma / lista.Count;
        Console.WriteLine("A média dos elementos da lista é: " + media);
    }
}
```

Java:

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class MediaElementosLista {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        List<Integer> lista = new ArrayList<>();
        System.out.print("Digite 5 números separados por espaço: ");
        for (int i = 0; i < 5; i++) {
            lista.add(scanner.nextInt());
        }
        int soma = 0;
        for (int elemento : lista) {
            soma += elemento;
        }

        double media = (double) soma / lista.size();
        System.out.println("A média dos elementos da lista é: " + media);
    }
}
```

Funções e Procedimentos

Exercício 1: Função para Calcular a Potência de um Número

C#:

```
using System;

class Program
{
    static double Potencia(double baseNum, double expoente)
    {
        return Math.Pow(baseNum, expoente);
    }

    static void Main()
    {
        Console.Write("Digite a base: ");
        double baseNum = double.Parse(Console.ReadLine());

        Console.Write("Digite o expoente: ");
        double expoente = double.Parse(Console.ReadLine());

        double resultado = Potencia(baseNum, expoente);
        Console.WriteLine("O resultado é: " + resultado);
    }
}
```

Java:

```
import java.util.Scanner;

public class Potencia {
    static double potencia(double baseNum, double expoente) {
        return Math.pow(baseNum, expoente);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite a base: ");
        double baseNum = scanner.nextDouble();

        System.out.print("Digite o expoente: ");
        double expoente = scanner.nextDouble();

        double resultado = potencia(baseNum, expoente);
        System.out.println("O resultado é: " + resultado);
    }
}
```

Funções e Procedimentos

Exercício 2: Procedimento para Exibir uma Mensagem

C#:

```
using System;

class Program
{
    static void ExibirMensagem(string mensagem)
    {
        Console.WriteLine(mensagem);
    }

    static void Main()
    {
        ExibirMensagem("Olá, mundo!");
    }
}
```

Java:

```
import java.util.Scanner;

public class ExibirMensagem {
    static void exibirMensagem(String mensagem) {
        System.out.println(mensagem);
    }

    public static void main(String[] args) {
        exibirMensagem("Olá, mundo!");
    }
}
```

Funções e Procedimentos

Exercício 3: Função para Verificar se um Número é Primo

C#:

```
using System;

class Program
{
    static bool EhPrimo(int num)
    {
        if (num <= 1)
        {
            return false;
        }

        for (int i = 2; i <= Math.Sqrt(num); i++)
        {
            if (num % i == 0)
            {
                return false;
            }
        }

        return true;
    }

    static void Main()
    {
        Console.Write("Digite um número: ");
        int num = int.Parse(Console.ReadLine());

        if (EhPrimo(num))
        {
            Console.WriteLine("O número é primo.");
        }
        else
        {
            Console.WriteLine("O número não é primo.");
        }
    }
}
```

Java:

```
import java.util.Scanner;

public class VerificarNumeroPrimo {
    static boolean ehPrimo(int num) {
        if (num <= 1) {
            return false;
        }

        for (int i = 2; i <= Math.sqrt(num); i++) {
            if (num % i == 0) {
                return false;
            }
        }

        return true;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um número: ");
        int num = scanner.nextInt();

        if (ehPrimo(num)) {
            System.out.println("O número é primo.");
        } else {
            System.out.println("O número não é primo.");
        }
    }
}
```

Funções e Procedimentos

Exercício 4: Procedimento para Imprimir um Triângulo

C#:

```
using System;

class Program
{
    static void ImprimirTriangulo(int linhas)
    {
        for (int i = 1; i <= linhas; i++)
        {
            for (int j = 1; j <= i; j++)
            {
                Console.Write("* ");
            }
            Console.WriteLine();
        }
    }
    static void Main()
    {
        Console.Write("Digite o número de linhas do triângulo: ");
        int linhas = int.Parse(Console.ReadLine());

        ImprimirTriangulo(linhas);
    }
}
```

Java:

```
import java.util.Scanner;

public class ImprimirTriangulo {
    static void imprimirTriangulo(int linhas) {
        for (int i = 1; i <= linhas; i++) {
            for (int j = 1; j <= i; j++) {
                System.out.print("* ");
            }
            System.out.println();
        }
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite o número de linhas do triângulo: ");
        int linhas = scanner.nextInt();

        imprimirTriangulo(linhas);
    }
}
```

Funções e Procedimentos

Exercício 5: Função para Calcular o Fatorial de um Número

C#:

```
using System;

class Program
{
    static int Fatorial(int num)
    {
        if (num == 0 || num == 1)
        {
            return 1;
        }

        int resultado = 1;
        for (int i = 2; i <= num; i++)
        {
            resultado *= i;
        }

        return resultado;
    }

    static void Main()
    {
        Console.Write("Digite um número: ");
        int num = int.Parse(Console.ReadLine());

        int fatorial = Fatorial(num);
        Console.WriteLine("O fatorial de " + num + " é: " + fatorial);
    }
}
```

Java:

```
import java.util.Scanner;

public class CalcularFatorial {
    static int fatorial(int num) {
        if (num == 0 || num == 1) {
            return 1;
        }

        int resultado = 1;
        for (int i = 2; i <= num; i++) {
            resultado *= i;
        }

        return resultado;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um número: ");
        int num = scanner.nextInt();

        int fatorial = fatorial(num);
        System.out.println("O fatorial de " + num + " é: " + fatorial);
    }
}
```