Introdução

Seja bem-vindo ao ebook de 200+ exercícios de Python! Com este material você irá se preparar para o mercado de trabalho e polir muito seu conhecimento em Python.

Preparei exercícios que são similares aos problemas que os desenvolvedores vivenciam nas empresas, então a grande maioria deles tem este foco no mercado de trabalho.

Os que não tem, visam trabalhar com partes que os programadores costumam ter dificuldade em Python, como conceitos mais avançados que utilizados na hora certa podem ser úteis demais.

Cada capítulo trata de um recurso de forma 'macro', sendo dividido em seções para explorar o recurso ao máximo. Como por exemplo o capítulo 4, que trabalha com controle de fluxo, ele tem as seções:

- If, else e else if;
- POO;
- Loops;

Onde os exercícios têm normalmente dificuldade progressiva, para você se sentir desafiado.

> Quero te convidar também a conhecer meu **Curso Completo de Python**, que será um ótimo material aliado a este para você dominar a linguagem.

E para você que está lendo este ebook há uma condição especial, clique no link acima e veja como o curso é completo e como ele vai te ajudar a programar melhor em Python.

Todos os nossos cursos possuem valores acessíveis, suporte para dúvidas, certificado de conclusão, exercícios e projetos. Recursos que vão amplificar seu aprendizado.

Quer conteúdo gratuito e de qualidade de programação? Se inscreva no meu canal de YouTube e me siga no Instagram: @horadecodar =)

Dada a introdução, vamos prosseguir?

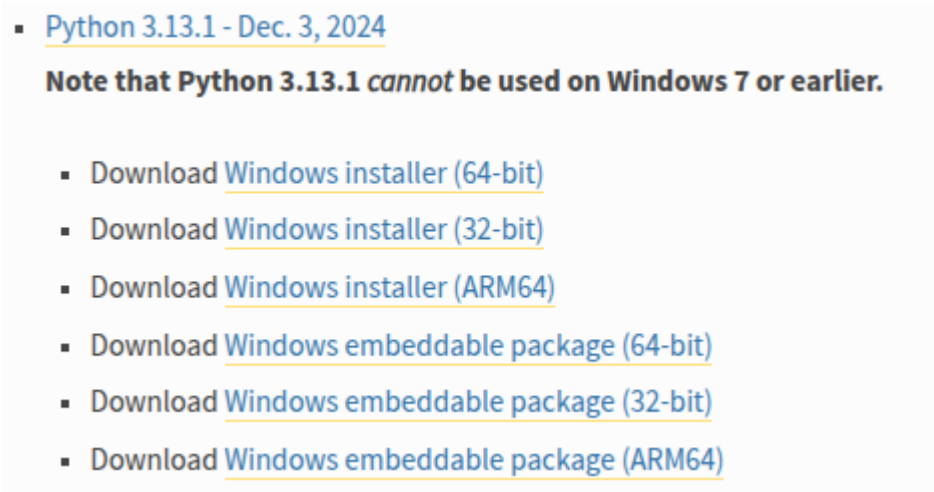
Capítulo 1: Configuração do Ambiente de Desenvolvimento

Se você já tem o Python instalado, passe para o segundo capítulo e comece os exercícios! =)

Instalando o Python no Windows

Passo 1: Baixar o Python

1. Acesse o site oficial do Python: https://www.python.org/
2. Escolha a versão do Python que você deseja instalar.
3. Para um ambiente moderno e sem bugs, eu indico o Python 3.13 ou a versão que estiver em LTS.
4. Clique no link de download correspondente à sua versão do Windows (64 bits).



Passo 2: Instalar o Python

1. Após o download, abra o instalador.
2. Siga as instruções do instalador, aceitando os termos de uso e selecionando a pasta de instalação padrão ou uma personalizada.
3. Certifique-se de que a opção "Add python.exe to PATH" esteja marcada, pois isso define a variável de ambiente necessária para que o sistema reconheça o python.
4. Conclua a instalação clicando em "Install" e depois em "Finish".

Obs: Você pode deixar todas as opções marcadas, elas vão deixar o seu ambiente mais completo.

Passo 3: Verificar a Instalação do Python

1. Abra o Prompt de Comando:
2. Pressione Win + R, digite cmd e pressione Enter.
3. Digite o comando abaixo para verificar a versão instalada do Python:

```
python –version ou python3 --version
```

Desta maneira:

```
C:\Users\mathe>python --version
Python 3.13.0
```

Se a instalação estiver correta, você verá algo como:

> python 3.13.0

Instalando o Visual Studio Code (VS Code)

Passo 1: Baixar o VS Code

1. Acesse o site oficial do Visual Studio Code: https://code.visualstudio.com/
2. Clique em Download for Windows para baixar a versão do instalador para Windows.

Passo 2: Instalar o VS Code

1. Após o download, execute o instalador do VS Code.
2. Marque a opção "Add to PATH" durante a instalação para que o VS Code possa ser aberto diretamente pelo terminal.
3. Siga as instruções e conclua a instalação clicando em "Install".

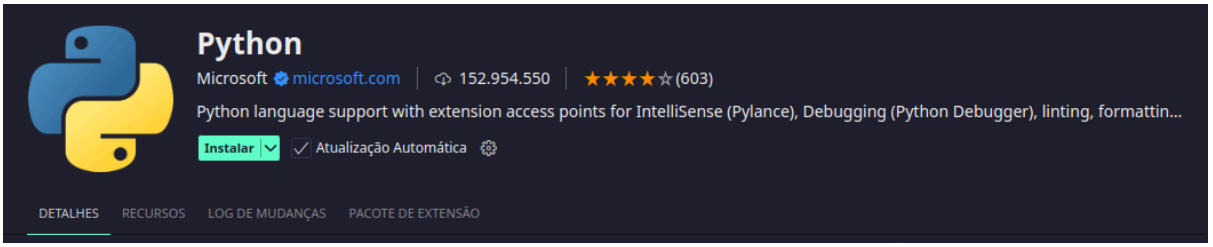
Configurando o Ambiente Python no VS Code

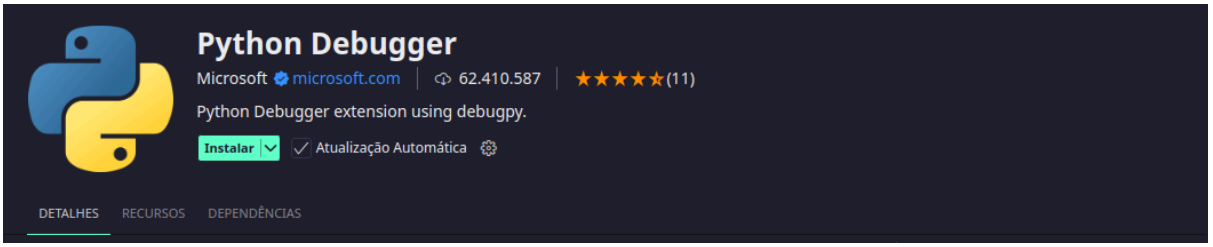
Passo 1: Abrir o VS Code

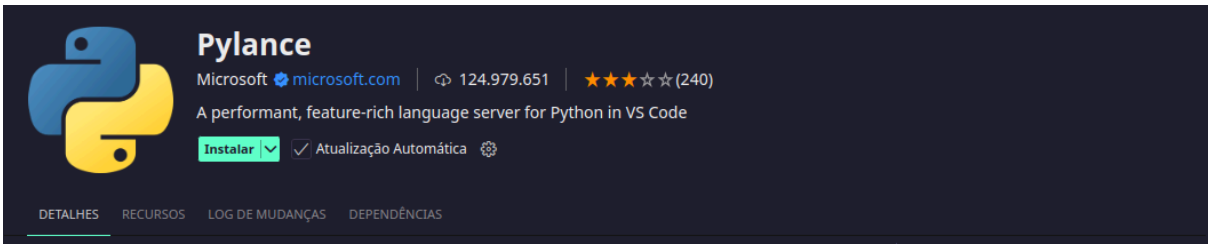
1. Abra o Visual Studio Code através do atalho no desktop ou pelo menu Iniciar.
2. Vá até a aba Extensões clicando no ícone de blocos no lado esquerdo do editor ou usando o atalho Ctrl + Shift + X.

Passo 2: Instalar as Extensões

1. Na aba de extensões, pesquise por Python, Python Debugger e Pylance.
2. Clique no resultado e, em seguida, clique no botão Instalar.







Essas são as extensões.

Esta extensão instala automaticamente um conjunto de ferramentas essenciais para desenvolvimento Python no VS Code, incluindo:

- Python
- Python Debugger
- Pylance

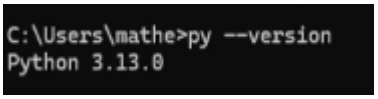
Isso vai fazer com que você consiga executar o Python pelo VS Code e ter highlight da linguagem, o que facilita as coisas.

Passo 3: Verificar a Instalação da Extensão

1. Após a instalação, reinicie o VS Code.
2. Para verificar se tudo está instalado corretamente, abra um novo terminal no VS Code (Ctrl + ~) e digite:

```
py -version
```

Se a instalação do Python e da extensão estiver correta, a versão do Python instalada será exibida no terminal.



Criando e Executando um Programa Python no VS Code

Passo 1: Criar um Novo Projeto Python

1. Crie uma pasta no seu computador

2. Abra o VS Code nela

Passo 2: Escrever o Código Python

Após a criação do projeto, crie um novo arquivo chamado Main.py na pasta.

Adicione o código Python abaixo ao arquivo Main.py:

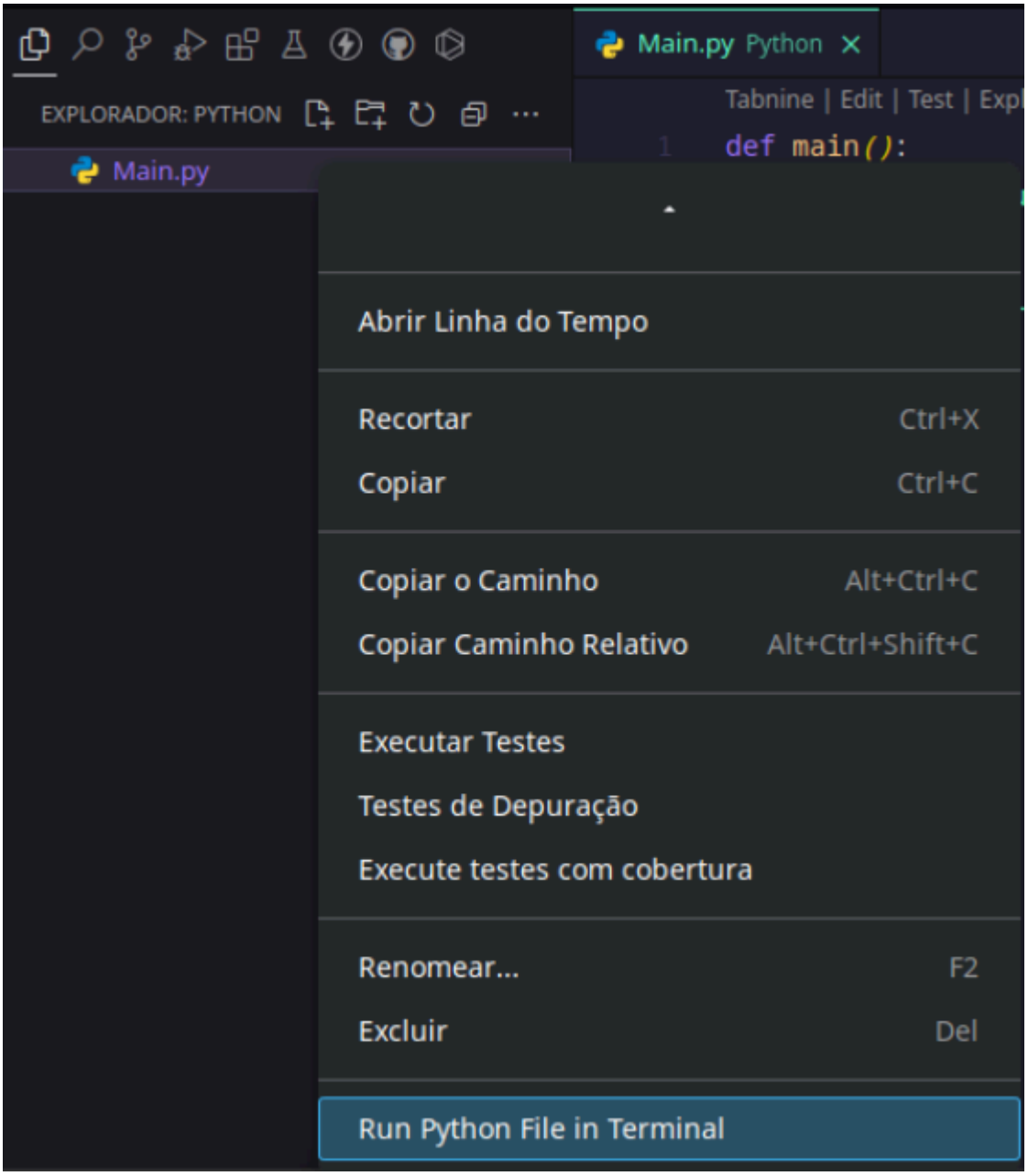
```
def main():
    print("Olá, Mundo!")

if __name__ == "__main__":
    main()
```

Passo 3: Compilar e Executar o Programa

Para compilar e executar, basta clicar no arquivo com o botão direito do mouse e escolher a opção Run Python

Veja:



Agora você já está apto a utilizar o Python no seu computador, vamos aos exercícios!

Nota importante

1) Aqui no nosso ebook eu fiz os exercícios separadamente por classes. Mas você pode unir vários em uma classe só, para não ficar tão burocrático.

Por exemplo: Divida todos os exercícios de cada capítulo em uma classe.

Os arquivos de conferência no GitHub estão organizados desta forma também.

2) Alguns exercícios solicitam os dados do usuário, utilize a classe Scanner para obter os dados.

> Quero te convidar também a conhecer meu **Curso Completo de Python**, que vai te fazer superar qualquer dificuldade nos exercícios deste ebook, e ainda está numa condição especial para os leitores deste material, aproveite! =)

Capítulo 2: Introdução ao Python

Neste capítulo, você dará os primeiros passos na linguagem de programação Python, uma das mais populares e versáteis do mundo. A jornada começa com a compreensão dos fundamentos que são essenciais para qualquer programador que deseja dominar Python.

O que você vai aprender?

Você será introduzido aos conceitos básicos, como a criação do seu primeiro programa, a importância dos comentários no código e a declaração e manipulação de variáveis. Além disso, abordaremos operações aritméticas, a utilização de constantes e como realizar a leitura e a exibição de dados no console.

Desafios que você enfrentará:

Neste capítulo, os desafios serão focados em entender e aplicar conceitos fundamentais. Você aprenderá a lidar com diferentes tipos de dados e variáveis, a realizar operações básicas e a formatar a saída de dados de forma adequada. Alguns exercícios também irão desafiá-lo a entender as diferenças entre variáveis locais e globais, além de aprender a trabalhar com strings e a manipular a entrada de dados do usuário.

Ao concluir os exercícios deste capítulo, você estará familiarizado com a estrutura básica de um programa em Python e entenderá como os diferentes tipos de dados e operações funcionam na prática. Esta base sólida é essencial para enfrentar desafios mais complexos nos próximos capítulos. Prepare-se para começar sua jornada na programação Python!

Exercício 1: Primeiros Passos

Crie um programa Python que exiba a mensagem "Olá, Mundo!" no console. Em seguida, modifique o programa para exibir seu nome.

Código de solução:

```
# Exibindo "Olá, Mundo!" no console
print("Olá, Mundo!")

# Modificando para exibir o nome
print("Meu nome é Matheus.")
```

Neste exercício, você cria um programa simples que exibe uma mensagem no console usando a função print. Primeiro, a mensagem "Olá, Mundo!" é exibida, e depois é modificada para exibir seu nome. Este exercício introduz a estrutura básica de um programa Python e a função de saída no console.

Exercício 2: Comentários no Código

Escreva um programa Python que contenha três tipos de comentários: comentário de linha, comentário de bloco e docstrings. Explique brevemente o uso de cada um desses comentários dentro do código.

Código de solução:

```
def comentarios():
    # Este é um comentário de linha, usado para comentar uma única linha

    """
    Este é um comentário de bloco,
    pode ser usado para comentar várias linhas
    """

    '''
    Docstring: usada para documentar funções, classes
    ou módulos, e pode ser acessada através do atributo
    especial __doc__
    '''

    print("Comentários adicionados ao código!")

if __name__ == "__main__":
    comentarios()
```

Explicação: Este exercício demonstra o uso de três tipos de comentários em Python:

Comentário de linha (#): Usado para adicionar anotações em uma única linha.
Comentário de bloco (""" ... """ ou ''' ... '''): Utilizado para comentar múltiplas linhas de código.
Docstring (""" ... """ ou ''' ... '''): Utilizada para documentar funções, classes ou módulos. As docstrings são acessíveis durante a execução do programa por meio do atributo especial __doc__.

Exercício 3: Variáveis e Tipos de Dados

Crie um programa que declare e inicialize variáveis de todos os tipos primitivos em Python (int, float, str, bool). Exiba o valor de cada variável no console.

Código de solução:

```
def tipos_de_dados():
    idade = 25          # int: número inteiro
    altura = 1.75       # float: número com ponto decimal
    inicial = 'M'       # str: caractere (Python não possui tipo char,
usamos string de um caractere)
    estudante = True    # bool: valor booleano (True ou False)

    print("Idade:", idade)
    print("Altura:", altura)
    print("Inicial do nome:", inicial)
    print("É estudante?", estudante)

if __name__ == "__main__":
    tipos_de_dados()
```

Explicação: Neste exercício, você declara variáveis de diferentes tipos primitivos em Python:

int: Usado para números inteiros.
float: Usado para números decimais.
str: Usado para strings
bool: Usado para valores booleanos (True ou False).
O programa exibe os valores dessas variáveis no console usando a função print. Este exercício ajuda a compreender a utilização de diferentes tipos de dados em Python.

Exercício 4: Conversão de Tipos

Escreva um programa que converta um valor float em int e outro valor int em float. Exiba os resultados das conversões e explique a diferença entre conversão explícita e implícita.

Código de solução:

```
def conversao_de_tipos():
    valor_float = 9.99
    valor_int = int(valor_float)  # Conversão explícita de float para
int (trunca o valor decimal)

    numero = 10
```

```
    numero_convertido = float(numero)  # Conversão implícita de int para
float

    print("Valor float:", valor_float)
    print("Valor convertido para int:", valor_int)
    print("Número int:", numero)
    print("Número convertido para float:", numero_convertido)

if __name__ == "__main__":
    conversao_de_tipos()
```

Explicação: Este exercício demonstra a conversão de tipos em Python:
Conversão explícita: De float para int usando a função int(). Nesse caso, o valor decimal é truncado (a parte decimal é descartada).

Conversão implícita: De int para float. Em Python, essa conversão ocorre automaticamente durante operações que envolvem tipos mistos, sem perda de informação.

Este exercício reforça o conceito de conversão de tipos em Python e mostra como a conversão explícita pode resultar em perda de precisão, enquanto a conversão implícita preserva o valor original.

Exercício 5: Operações Aritméticas

Desenvolva um programa que declare duas variáveis inteiras e realize as operações de soma, subtração, multiplicação, divisão e módulo entre elas. Exiba os resultados de cada operação.

Código de solução:

```
def operacoes_aritmeticas():
    a = 10
    b = 3

    soma = a + b
    subtracao = a - b
    multiplicacao = a * b
    divisao = a // b  # Divisão inteira
    modulo = a % b

    print("Soma:", soma)
    print("Subtração:", subtracao)
    print("Multiplicação:", multiplicacao)
    print("Divisão:", divisao)
    print("Módulo:", modulo)
```

```
if __name__ == "__main__":
    operacoes_aritmeticas()
```

Explicação: Neste exercício, são realizadas operações aritméticas básicas em Python:

Soma (+): Adiciona os valores de a e b.
Subtração (-): Subtrai b de a.
Multiplicação (*): Multiplica os valores de a e b.
Divisão inteira (//): Realiza a divisão inteira, retornando apenas a parte inteira do quociente.
Módulo (%): Retorna o restante da divisão de a por b.

O programa exibe o resultado de cada operação no console. Este exercício ajuda a solidificar o entendimento das operações matemáticas em Python e sua sintaxe.

Exercício 6: Constantes

Crie um programa que utilize a palavra-chave para declarar uma constante que representa a velocidade da luz no vácuo. Tente alterar o valor da constante e observe o comportamento do Python.

Código de solução:

```
def constantes():
    VELOCIDADE_DA_LUZ = 299792458  # Velocidade da luz em metros por segundo

    print(f"Velocidade da luz: {VELOCIDADE_DA_LUZ} m/s")

    # Tentar alterar a constante (vai gerar um erro, pois em Python não há uma palavra-chave 'final')
    try:
        VELOCIDADE_DA_LUZ = 300000000  # Isso irá gerar um erro de execução
    except Exception as e:
        print(f"Erro ao tentar alterar a constante: {e}")

if __name__ == "__main__":
    constantes()
```

Explicação: Em Python, não existe uma palavra-chave como final para declarar constantes. Porém, por convenção, usamos letras maiúsculas para indicar que um valor é constante e não deve ser alterado.

Neste exercício:
Declaramos uma variável chamada VELOCIDADE_DA_LUZ que representa a velocidade da luz.
Tentamos alterar o valor dessa variável, mas em Python isso não gera um erro de compilação, pois Python não tem uma restrição formal de imutabilidade para variáveis. No entanto, a tentativa de alteração pode ser monitorada com um try-except para simular um erro caso o programador altere acidentalmente o valor da "constante".
Este exercício demonstra como em Python, embora não existam palavras-chave específicas para constantes, a convenção de nomes pode ser usada para sinalizar essa intenção.

Exercício 7: Entrada de Dados

Escreva um programa que leia um número inteiro e um número decimal do teclado e, em seguida, exiba a soma desses números no console.

Código de solução:

```
def entrada_de_dados():
    numero_int = int(input("Digite um número inteiro: "))
    numero_double = float(input("Digite um número decimal: "))

    soma = numero_int + numero_double
    print(f"A soma dos números é: {soma}")

if __name__ == "__main__":
    entrada_de_dados()
```

Explicação: Em Python, a leitura de dados do teclado é feita utilizando a função input(), que retorna o valor como uma string. Para convertê-lo para um número inteiro, usamos int(), e para um número decimal, usamos float().

Neste exercício:
O programa lê um número inteiro e um número decimal do usuário.
A soma desses dois valores é calculada e exibida no console.
Este exercício ajuda a entender como trabalhar com entrada de dados no Python e realizar operações com tipos diferentes de dados.

Exercício 8: Strings e Concatenação

Crie um programa que peça ao usuário para digitar seu nome e sobrenome. O programa deve exibir uma mensagem de boas-vindas concatenando o nome e o sobrenome do usuário.

Código de solução:

```
def boas_vindas():
    nome = input("Digite seu nome: ")
    sobrenome = input("Digite seu sobrenome: ")

    mensagem = f"Bem-vindo, {nome} {sobrenome}!"
    print(mensagem)

if __name__ == "__main__":
    boas_vindas()
```

Explicação: Em Python, usamos a função input() para ler a entrada do usuário. Para concatenar strings, é comum usar o f-string, que é uma forma simples e eficiente de incorporar variáveis dentro de strings.

Neste exercício:
O programa solicita que o usuário digite seu nome e sobrenome.
Em seguida, uma mensagem de boas-vindas é exibida, combinando essas informações em uma string.
A concatenação de strings é uma operação básica que permite combinar informações fornecidas pelo usuário, sendo fundamental em diversas situações de programação.

Exercício 9: Tipos de Variáveis

Escreva um programa que declare variáveis locais e globais (dentro de uma classe). Inicialize e exiba o valor de ambas as variáveis no console.

Código de solução:

```
class TiposDeVariaveis:
    # Variável global
    variavelGlobal = 10

    def mostrar_valores(self):
        # Variável local
        variavelLocal = 5

        print(f"Valor da variável global: {self.variavelGlobal}")
        print(f"Valor da variável local: {variavelLocal}")
```

```
if __name__ == "__main__":
    variaveis = TiposDeVariaveis()
    variaveis.mostrar_valores()
```

Explicação: Neste código em Python:

A variável variavelGlobal é declarada como uma variável global dentro da classe TiposDeVariaveis. Ela pode ser acessada por qualquer método da classe.
A variável variavelLocal é declarada dentro do método mostrar_valores, sendo local a esse método, ou seja, só pode ser acessada dentro dele.
O programa imprime os valores de ambas as variáveis no console usando print().

Exercício 10: Formatação de Saída

Desenvolva um programa que exiba o valor de uma variável double com duas casas decimais. Utilize formatação para garantir que o valor seja exibido corretamente.

Código de solução:

```
valor = 123.456789

# Exibe o valor formatado com duas casas decimais
print(f"Valor formatado: {valor:.2f}")
```

Explicação: Neste exercício, utilizamos f-strings para formatar a saída de uma variável float. A expressão {valor:.2f} garante que o valor seja exibido com exatamente duas casas decimais, que é um formato comum para representar valores monetários ou medidas precisas.

> Está com alguma dificuldade? Quero te convidar também a conhecer meu **Curso Completo de Python**, nele você dominará todos os conceitos dos exercícios deste ebook.

Capítulo 3: Operadores

Neste capítulo, você aprofundará seu conhecimento sobre operadores em Python, uma das ferramentas mais fundamentais na programação.

Operadores são símbolos especiais que nos permitem realizar operações em variáveis e valores, como somar números, comparar valores ou até tomar decisões baseadas em certas condições.

O que você vai aprender?

Você será introduzido aos diferentes tipos de operadores que Python oferece, incluindo:

- Operadores Aritméticos: usados para realizar operações matemáticas básicas como adição, subtração, multiplicação, divisão e módulo;
- Operadores de Comparação e Lógicos: que permitem comparar valores e construir expressões lógicas para controlar o fluxo do seu programa;
- Operadores de Atribuição e Incremento/Decremento: que facilitam a manipulação de valores armazenados em variáveis, permitindo atualizar esses valores de maneira eficiente.

Desafios que você enfrentará:

Neste capítulo, você enfrentará desafios que exigem:

- **Cálculos matemáticos**: aplicando operadores aritméticos para resolver problemas práticos, como calcular áreas, médias e realizar conversões;
- **Tomada de decisões**: utilizando operadores de comparação e lógicos para desenvolver programas que tomam decisões com base em condições específicas;
- **Manipulação de variáveis**: aprendendo a utilizar operadores de atribuição e incrementos para modificar e atualizar valores de variáveis de maneira controlada e eficiente;

Esses exercícios o ajudarão a entender como utilizar operadores para manipular dados e controlar o comportamento do seu programa.

Exercícios de Operadores Aritméticos

Exercício 11: Calculadora Simples

Crie um programa que leia dois números inteiros do usuário e exiba a soma, subtração, multiplicação, divisão e o módulo desses números no console.

Código de solução:

```
# Lê os dois números inteiros do usuário
num1 = int(input("Digite o primeiro número inteiro: "))
num2 = int(input("Digite o segundo número inteiro: "))
```

```
# Realiza as operações
soma = num1 + num2
subtracao = num1 - num2
multiplicacao = num1 * num2
divisao = num1 / num2 if num2 != 0 else "Erro: divisão por zero"
modulo = num1 % num2 if num2 != 0 else "Erro: divisão por zero"

# Exibe os resultados
print("Soma:", soma)
print("Subtração:", subtracao)
print("Multiplicação:", multiplicacao)
print("Divisão:", divisao)
print("Módulo:", modulo)
```

Explicação: Esse programa lê dois números inteiros do usuário e realiza operações de soma, subtração, multiplicação, divisão e módulo. Os resultados são exibidos no console, com verificação para evitar erro de divisão por zero.

Exercício 12: Média Aritmética

Escreva um programa que leia três números inteiros do usuário e calcule a média aritmética deles. Exiba o resultado no console.

Dica: Sempre que os exercícios pedirem para 'ler' algo, você deve utilizar a classe Scanner.

Código de solução:

```
# Lê três números inteiros do usuário
num1 = int(input("Digite o primeiro número inteiro: "))
num2 = int(input("Digite o segundo número inteiro: "))
num3 = int(input("Digite o terceiro número inteiro: "))

# Calcula a média aritmética
media = (num1 + num2 + num3) / 3.0

# Exibe o resultado
print(f"A média aritmética é: {media}")
```

Explicação: O programa solicita ao usuário que digite três números inteiros, calcula a média aritmética deles e exibe o resultado no console. A divisão por `3.0` garante que a média seja um número de ponto flutuante, preservando as casas decimais no resultado.

Exercício 13: Área de um Retângulo

Desenvolva um programa que leia a largura e a altura de um retângulo e calcule a área. Exiba o resultado no console.

Dica: area = largura x altura.

Código de solução:

```
# Lê a largura e a altura do retângulo
largura = float(input("Digite a largura do retângulo: "))
altura = float(input("Digite a altura do retângulo: "))

# Calcula a área
area = largura * altura

# Exibe o resultado
print(f"A área do retângulo é: {area}")
```

Explicação: O programa solicita ao usuário que insira a largura e a altura de um retângulo. Em seguida, calcula a área multiplicando esses dois valores e exibe o resultado no console.

Exercício 14: Conversão de Temperatura

Crie um programa que converta uma temperatura em graus Celsius para Fahrenheit. A fórmula de conversão é: F = (C * 9/5) + 32. Exiba o resultado no console.

Código de solução:

```
# Lê a temperatura em graus Celsius
celsius = float(input("Digite a temperatura em graus Celsius: "))

# Converte para Fahrenheit
fahrenheit = (celsius * 9/5) + 32

# Exibe o resultado
print(f"A temperatura em Fahrenheit é: {fahrenheit}")
```

Explicação: O programa solicita que o usuário insira a temperatura em graus Celsius. Em seguida, a fórmula de conversão é aplicada para calcular a temperatura equivalente em Fahrenheit, e o resultado é exibido no console.

Exercício 15: Potenciação

Escreva um programa que leia dois números inteiros do usuário e exiba o resultado da potenciação do primeiro número elevado ao segundo número (use o método Math.pow).

Dica: pow recebe dois argumentos, o primeiro a base e o segundo o expoente.

Código de solução:

```
# Programa que calcula a potência de um número

# Lê os números do usuário
base = int(input("Digite o número base: "))
expoente = int(input("Digite o expoente: "))

# Calcula o resultado da potenciação
resultado = pow(base, expoente)

# Exibe o resultado
print(f"O resultado de {base} elevado a {expoente} é: {resultado}")
```

Explicação: Neste programa, a função pow() é utilizada para calcular a potência de um número, onde o primeiro argumento é a base e o segundo é o expoente. O resultado é então exibido no console.

Exercícios Operadores de Comparação e Lógicos

Exercício 16: Comparação Simples

Crie um programa que leia dois números inteiros e exiba se o primeiro é maior, menor ou igual ao segundo.

Código de solução:

```
# Programa que compara dois números inteiros

# Lê os números do usuário
num1 = int(input("Digite o primeiro número inteiro: "))
num2 = int(input("Digite o segundo número inteiro: "))

# Compara os números e exibe o resultado
if num1 > num2:
```

```
    print("O primeiro número é maior que o segundo.")
elif num1 < num2:
    print("O primeiro número é menor que o segundo.")
else:
    print("Os dois números são iguais.")
```

Explicação: Este programa compara dois números inteiros fornecidos pelo usuário e exibe uma mensagem indicando se o primeiro número é maior, menor ou igual ao segundo. O if, elif e else são usados para verificar as diferentes condições de comparação.

Exercício 17: Verificação de Paridade

Escreva um programa que leia um número inteiro e exiba se ele é par ou ímpar.

Dica: Você pode utilizar a divisão de resto, com o operador %.

Código de solução:

```
# Programa que verifica se um número é par ou ímpar

# Lê um número inteiro do usuário
num = int(input("Digite um número inteiro: "))

# Verifica se o número é par ou ímpar
if num % 2 == 0:
    print("O número é par.")
else:
    print("O número é ímpar.")
```

Explicação: Este programa lê um número inteiro fornecido pelo usuário e verifica se ele é par ou ímpar usando o operador de módulo %. Se o resto da divisão do número por 2 for zero, o número é par; caso contrário, ele é ímpar.

Exercício 18: Maior de Três Números

Desenvolva um programa que leia três números inteiros e exiba o maior deles.

Código de solução:

```
# Programa que lê três números inteiros e exibe o maior deles

```

```
# Lê os três números inteiros do usuário
num1 = int(input("Digite o primeiro número inteiro: "))
num2 = int(input("Digite o segundo número inteiro: "))
num3 = int(input("Digite o terceiro número inteiro: "))

# Determina o maior número
maior = num1

if num2 > maior:
    maior = num2

if num3 > maior:
    maior = num3

# Exibe o maior número
print("O maior número é:", maior)
```

Explicação: Neste programa, o usuário insere três números inteiros, e o código determina qual deles é o maior. Utilizamos uma estrutura condicional simples para comparar os números e armazenar o maior valor na variável maior, que é então exibido ao usuário.

Exercício 19: Elegibilidade para Votação

Crie um programa que leia a idade de uma pessoa e verifique se ela é elegível para votar (idade igual ou superior a 18 anos).

Código de solução:

```
# Programa que verifica se uma pessoa é elegível para votar

# Lê a idade da pessoa
idade = int(input("Digite a sua idade: "))

# Verifica se a pessoa é elegível para votar
if idade >= 18:
    print("Você é elegível para votar.")
else:
    print("Você não é elegível para votar.")
```

Explicação: O programa solicita ao usuário que digite sua idade. Se a idade for igual ou superior a 18 anos, o programa informa que a pessoa é elegível para votar; caso contrário, informa que a pessoa não é elegível.

Exercício 20: Verificação de Intervalo

Escreva um programa que leia um número inteiro e verifique se ele está entre 10 e 20 (inclusive). Exiba uma mensagem informando se o número está dentro ou fora do intervalo.

Código de solução:

```
# Programa que verifica se um número está entre 10 e 20

# Lê o número inteiro
num = int(input("Digite um número inteiro: "))

# Verifica se o número está dentro do intervalo
if 10 <= num <= 20:
    print("O número está dentro do intervalo.")
else:
    print("O número está fora do intervalo.")
```

Explicação: O programa solicita que o usuário insira um número inteiro. Em seguida, verifica se o número está entre 10 e 20 (inclusive) e exibe a mensagem correspondente.

Exercício 21: Comparação de Strings

Desenvolva um programa que leia duas strings do usuário e verifique se elas são iguais. Exiba uma mensagem informando o resultado da comparação.

Código de solução:

```
# Programa que compara duas strings

# Lê as duas strings
string1 = input("Digite a primeira string: ")
string2 = input("Digite a segunda string: ")

# Verifica se as strings são iguais
if string1 == string2:
    print("As strings são iguais.")
else:
    print("As strings são diferentes.")
```

Explicação: Este programa solicita que o usuário insira duas strings. Ele compara as duas strings utilizando o operador ==, que verifica se elas são iguais, e exibe a mensagem correspondente ao resultado da comparação.

Exercício 22: Operadores Lógicos AND e OR

Crie um programa que leia três números inteiros e verifique se pelo menos um deles é maior que 10 (usando o operador ||). Em seguida, verifique se todos são maiores que 10 (usando o operador &&).

Código de solução:

```
# Programa que verifica condições usando operadores lógicos

# Lê os três números inteiros
num1 = int(input("Digite o primeiro número inteiro: "))
num2 = int(input("Digite o segundo número inteiro: "))
num3 = int(input("Digite o terceiro número inteiro: "))

# Verifica se pelo menos um dos números é maior que 10
if num1 > 10 or num2 > 10 or num3 > 10:
    print("Pelo menos um dos números é maior que 10.")
else:
    print("Nenhum dos números é maior que 10.")

# Verifica se todos os números são maiores que 10
if num1 > 10 and num2 > 10 and num3 > 10:
    print("Todos os números são maiores que 10.")
else:
    print("Nem todos os números são maiores que 10.")
```

Explicação: Este programa lê três números inteiros do usuário e utiliza operadores lógicos para realizar duas verificações:
Usando o operador or, ele verifica se pelo menos um dos números é maior que 10.
Usando o operador and, ele verifica se todos os números são maiores que 10.
O programa então exibe a mensagem apropriada com base nos resultados dessas verificações.

Exercício 23: Verificação de Maioria

Escreva um programa que leia a idade de três pessoas e verifique se pelo menos duas delas são maiores de idade (18 anos ou mais).

Código de solução:

```
# Programa que verifica se pelo menos duas pessoas são maiores de idade

# Lê as idades das três pessoas
```

```
idade1 = int(input("Digite a idade da primeira pessoa: "))
idade2 = int(input("Digite a idade da segunda pessoa: "))
idade3 = int(input("Digite a idade da terceira pessoa: "))

# Conta quantas pessoas são maiores de idade
maioridade = 0

# Verifica se cada pessoa tem 18 anos ou mais
if idade1 >= 18:
    maioridade += 1
if idade2 >= 18:
    maioridade += 1
if idade3 >= 18:
    maioridade += 1

# Verifica se pelo menos duas pessoas são maiores de idade
if maioridade >= 2:
    print("Pelo menos duas pessoas são maiores de idade.")
else:
    print("Menos de duas pessoas são maiores de idade.")
```

Explicação: Este código lê as idades de três pessoas e conta quantas delas são maiores de idade (18 anos ou mais). Se pelo menos duas pessoas forem maiores de idade, ele imprime a mensagem correspondente. Caso contrário, ele informa que menos de duas pessoas são maiores de idade.

Exercício 24: Ano Bissexto

Crie um programa que leia um ano e verifique se ele é bissexto. Um ano é bissexto se for divisível por 4, mas não por 100, exceto se for divisível por 400.

Código de solução:

```
# Programa para verificar se um ano é bissexto

# Lê o ano
ano = int(input("Digite um ano: "))

# Verifica se o ano é bissexto
if (ano % 4 == 0 and ano % 100 != 0) or (ano % 400 == 0):
    print("O ano é bissexto.")
else:
    print("O ano não é bissexto.")
```

Explicação: Este programa lê um ano fornecido pelo usuário e verifica se ele é bissexto com base nas seguintes regras:

O ano é divisível por 4, mas não por 100, ou
O ano é divisível por 400.
Se o ano atender a uma dessas condições, ele será bissexto; caso contrário, não será.

Exercício 25: Verificação de Números Positivos

Escreva um programa que leia três números inteiros e verifique se pelo menos dois deles são positivos.

Código de solução:

```
# Programa para verificar se pelo menos dois números são positivos

# Lê três números inteiros
num1 = int(input("Digite o primeiro número inteiro: "))
num2 = int(input("Digite o segundo número inteiro: "))
num3 = int(input("Digite o terceiro número inteiro: "))

# Contador de números positivos
positivos = 0

# Verifica se os números são positivos
if num1 > 0:
    positivos += 1
if num2 > 0:
    positivos += 1
if num3 > 0:
    positivos += 1

# Verifica se pelo menos dois números são positivos
if positivos >= 2:
    print("Pelo menos dois dos números são positivos.")
else:
    print("Menos de dois dos números são positivos.")
```

Explicação: Este programa lê três números inteiros e conta quantos deles são positivos. Se pelo menos dois números forem positivos, ele imprime a mensagem correspondente; caso contrário, informa que menos de dois números são positivos.

Operadores de Atribuição e Incremento/Decremento

Exercício 26: Atribuição Simples

Enunciado: Crie um programa que declare uma variável inteira, atribua um valor a ela e, em seguida, modifique o valor utilizando os operadores de atribuição (+=, -=, *=, /=, %=). Exiba o resultado após cada operação.

Código de solução:

```
# Programa para demonstrar operadores de atribuição

# Declara a variável e atribui um valor
valor = 10

# Utiliza os operadores de atribuição e exibe o resultado após cada
operação
valor += 5
print("Após valor += 5:", valor)

valor -= 3
print("Após valor -= 3:", valor)

valor *= 2
print("Após valor *= 2:", valor)

valor /= 4
print("Após valor /= 4:", valor)

valor %= 3
print("Após valor %= 3:", valor)
```

Explicação: Este programa começa com uma variável `valor` inicializada com 10. Em seguida, são aplicados diferentes operadores de atribuição (`+=`, `-=`, `*=`, `/=`, `%=`), e após cada operação, o novo valor da variável é impresso.

Exercício 27: Incremento e Decremento

Enunciado: Escreva um programa que declare uma variável inteira, aplique o operador de incremento (usando += 1) e decremento (usando -= 1), e exiba o valor da variável antes e depois de cada operação.

Código de solução:

```
# Programa para demonstrar incremento e decremento

# Declara a variável
numero = 15

# Exibe o valor inicial
print("Valor inicial:", numero)

# Incremento
numero += 1
print("Após incremento (numero++):", numero)

# Decremento
numero -= 1
print("Após decremento (numero--):", numero)
```

Explicação: Neste programa, começamos com a variável numero igual a 15. Em seguida, o valor de numero é incrementado e depois decrementado, e o valor da variável é exibido antes e após cada operação. O comportamento do incremento e decremento é simulado com += 1 e -= 1.

Exercício 28: Soma Acumulada

Enunciado: Desenvolva um programa que leia cinco números inteiros do usuário, um por vez, e acumule a soma deles usando o operador de atribuição +=. Exiba o total acumulado ao final.

Dica: Utilize o Scanner para pedir os dados, e você pode utilizar um loop for para repetir a solicitação de dados cinco vezes.

Código de solução:

```
# Programa para somar cinco números inteiros fornecidos pelo usuário

soma_acumulada = 0

for i in range(1, 6):
    valor_digitado = int(input(f"Digite o {i}º número inteiro: "))
    soma_acumulada += valor_digitado
```

```
print(f"Soma acumulada: {soma_acumulada}")
```

Explicação: Este programa lê cinco números inteiros fornecidos pelo usuário, um por vez, e acumula a soma deles utilizando o operador de atribuição +=. O total acumulado é exibido ao final.

A principal diferença é que, em Python, usamos input() para capturar dados do usuário e int() para converter a entrada em número inteiro.

Exercício 29: Pré-incremento e Pós-incremento

Enunciado: Crie um programa que demonstre a diferença entre o pré-incremento (++variavel) e o pós-incremento (variavel++). Utilize exemplos práticos e exiba os resultados no console.

Código de solução:

```
# Programa para demonstrar a diferença entre pré-incremento e
pós-incremento

valor_pre_pos = 10

print(f"Valor inicial: {valor_pre_pos}")

# Pré-incremento
pre_incremento = valor_pre_pos + 1
print(f"Ao aplicar pré-incremento (++valor): {pre_incremento}")

# Pós-incremento
pos_incremento = valor_pre_pos
valor_pre_pos += 1
print(f"Ao aplicar pós-incremento (valor++): {pos_incremento}")

print(f"Valor final após pós-incremento: {valor_pre_pos}")
```

Explicação: Este programa mostra a diferença entre o pré-incremento e o pós-incremento:
No pré-incremento, o valor da variável é incrementado antes de ser usado, por isso o incremento ocorre antes da atribuição.
No pós-incremento, o valor da variável é utilizado primeiro e, em seguida, o incremento é aplicado.
A variável valor_pre_pos é inicialmente 10, e as operações demonstram como o valor muda com cada tipo de incremento.

Exercício 30: Operadores Compostos

Enunciado: Escreva um programa que leia dois números inteiros do usuário e aplique operadores compostos (e.g., +=, -=, *=, /=, %=) para modificar o valor da primeira variável em relação à segunda. Exiba o resultado após cada operação.

Dica: Use o Scanner para receber os números.

Código de solução:

```
# Programa para aplicar operadores compostos em dois números inteiros

# Leitura dos números
valor1 = int(input("Digite o primeiro número inteiro: "))
valor2 = int(input("Digite o segundo número inteiro: "))

# Operações com operadores compostos
valor1 += valor2
print(f"Apos valor1 += valor2: {valor1}")

valor1 -= valor2
print(f"Apos valor1 -= valor2: {valor1}")

valor1 *= valor2
print(f"Apos valor1 *= valor2: {valor1}")

valor1 /= valor2
print(f"Apos valor1 /= valor2: {valor1}")

valor1 %= valor2
print(f"Apos valor1 %= valor2: {valor1}")
```

Explicação: Este programa lê dois números inteiros fornecidos pelo usuário e aplica operadores compostos para modificar o valor de valor1 em relação a valor2. A cada operação, o novo valor de valor1 é exibido no console.

Conclusão do Capítulo 3

Neste capítulo, você explorou uma variedade de operadores em Python, que são fundamentais para manipular e controlar dados em seus programas. Os exercícios foram projetados para reforçar o entendimento dos seguintes conceitos:

Operadores Aritméticos: Você aprendeu a realizar operações matemáticas básicas, como adição, subtração, multiplicação, divisão e módulo, além de trabalhar com expressões mais complexas, como potenciação e cálculo de médias.

Operadores de Comparação e Lógicos: Você praticou a comparação de valores e a construção de expressões lógicas que permitem tomar decisões baseadas em condições específicas. Isso incluiu o uso de operadores de igualdade, maior/menor, e combinações lógicas utilizando AND, OR, e NOT.

Operadores de Atribuição e Incremento/Decremento: Você viu como modificar valores de variáveis de forma eficiente utilizando operadores compostos e incrementos/decrementos, entendendo a diferença entre pré-incremento e pós-incremento.

Ao dominar esses operadores, você desenvolveu habilidades essenciais para controlar o fluxo de seus programas, realizar cálculos e manipular dados de forma eficaz.

Porém, isso é apenas o começo, temos muito o que explorar no universo de Python através de exercícios! Vejo você no próximo capítulo.

Capítulo 4: Controle de Fluxo

Neste capítulo, você irá aprofundar seu conhecimento sobre o controle de fluxo em Python, um dos aspectos mais importantes da programação. O controle de fluxo permite que você tome decisões, repita ações e altere o comportamento do seu programa com base em diferentes condições e cenários.

O que você vai aprender?

- Estruturas Condicionais (if, elif, else): Você começará com as estruturas condicionais, que são fundamentais para a tomada de decisões em seus programas. Essas estruturas permitem que seu código execute diferentes blocos de instruções com base em condições específicas. Em Python, o if é utilizado para verificar uma condição, o elif para testar outras condições caso a primeira falhe, e o else para executar uma ação quando nenhuma das condições anteriores for atendida.

- Switch Case: Em Python, não existe a estrutura switch case como em outras linguagens. No entanto, você aprenderá a utilizar alternativas como dicionários ou a estrutura if-elif-else para criar um comportamento similar. Isso permite que você avalie uma única variável contra várias opções possíveis de maneira mais legível e organizada.

- Laços de Repetição (for, while): Finalmente, você explorará os laços de repetição, que permitem que você execute um bloco de código várias vezes, seja um número específico de vezes ou até que uma determinada condição seja atendida. Em

Python, você aprenderá a usar os laços for (que itera sobre uma sequência de valores) e while (que executa o código enquanto a condição for verdadeira). Ao contrário de outras linguagens, Python não possui o laço do-while, mas o comportamento pode ser simulado facilmente com o laço while.

Desafios que você enfrentará

Neste capítulo, os desafios incluirão desde a criação de simples tomadas de decisão, como verificar a validade de uma condição, até o uso de laços para repetir ações complexas. Você será desafiado a implementar soluções que exigem um pensamento lógico robusto e a utilização eficiente das estruturas de controle de fluxo em Python.

Com a conclusão deste capítulo, você terá uma compreensão sólida sobre como controlar o comportamento dos seus programas de forma dinâmica e eficiente, o que é essencial para resolver problemas de programação no mundo real. Prepare-se para aplicar essas ferramentas poderosas em uma variedade de situações que encontrará no desenvolvimento de software com Python.

Estruturas Condicionais (if, else if, else)

Exercício 31: Verificação de Positivo ou Negativo

Enunciado: Escreva um programa que leia um número inteiro e verifique se ele é positivo, negativo ou zero. Exiba uma mensagem apropriada para cada caso.

Código de solução:

```
# Programa para verificar se um número é positivo, negativo ou zero

# Solicita ao usuário um número inteiro
numero = int(input("Digite um número inteiro: "))

# Verifica se o número é positivo, negativo ou zero
if numero > 0:
    print("O número é positivo.")
elif numero < 0:
    print("O número é negativo.")
else:
    print("O número é zero.")
```

Explicação: O programa lê um número inteiro do usuário e, utilizando uma estrutura condicional if-elif-else, verifica se o número é positivo, negativo ou zero, exibindo a mensagem correspondente a cada caso.

Exercício 32: Par ou Ímpar

Enunciado: Crie um programa que leia um número inteiro e exiba se o número é par ou ímpar.

Código de solução:

```
# Programa para verificar se um número é par ou ímpar

# Solicita ao usuário um número inteiro
numero = int(input("Digite um número inteiro: "))

# Verifica se o número é par ou ímpar
if numero % 2 == 0:
    print("O número é par.")
else:
    print("O número é ímpar.")
```

Explicação: O programa lê um número inteiro do usuário e utiliza o operador módulo (%) para verificar se o número é divisível por 2. Se o resultado for zero, o número é par; caso contrário, ele é ímpar. O programa então exibe a mensagem correspondente.

Exercício 33: Cálculo de Desconto

Enunciado: Desenvolva um programa que leia o valor de uma compra e aplique um desconto de 10% se o valor for superior a R$100,00. Exiba o valor final com ou sem desconto.

Código de solução:

```
# Programa para aplicar desconto em uma compra

# Solicita ao usuário o valor da compra
valor_compra = float(input("Digite o valor da compra: R$ "))

# Verifica se o valor é superior a R$100,00 e aplica o desconto
if valor_compra > 100.00:
    valor_compra *= 0.90  # Aplicando desconto de 10%
    print(f"Desconto aplicado! Valor final: R$ {valor_compra:.2f}")
```

```
else:
    print(f"Sem desconto. Valor final: R$ {valor_compra:.2f}")
```

Explicação: O programa lê o valor da compra e verifica se é superior a R$100,00. Se for, aplica um desconto de 10% e exibe o valor final com o desconto. Caso contrário, o valor original é mantido e exibido. O valor é formatado com duas casas decimais para uma apresentação mais clara.

Exercício 34: Verificação de Maioridade

Enunciado: Escreva um programa que leia a idade de uma pessoa e exiba uma mensagem informando se ela é menor de idade (menor que 18 anos), maior de idade (18 anos ou mais) ou idosa (60 anos ou mais).

Código de solução:

```
# Programa para verificar a faixa etária

# Solicita ao usuário a idade
idade = int(input("Digite a sua idade: "))

# Verifica a faixa etária e exibe a mensagem correspondente
if idade >= 60:
    print("Você é idoso.")
elif idade >= 18:
    print("Você é maior de idade.")
else:
    print("Você é menor de idade.")
```

Explicação: O programa lê a idade do usuário e verifica, usando estruturas condicionais, se ele é menor de idade (menor que 18 anos), maior de idade (18 anos ou mais) ou idoso (60 anos ou mais). A mensagem apropriada é exibida com base na condição atendida.

Exercício 35: Classificação de Notas

Enunciado: Crie um programa que leia uma nota de 0 a 100 e exiba uma mensagem de aprovação se a nota for maior ou igual a 60. Caso contrário, exiba uma mensagem de reprovação.

Código de solução:

```python
# Programa para verificar a aprovação ou reprovação com base na nota

# Solicita ao usuário a nota
nota = int(input("Digite a sua nota (0 a 100): "))

# Verifica se a nota é suficiente para aprovação
if nota >= 60:
    print("Você está aprovado.")
else:
    print("Você está reprovado.")
```

Explicação: O programa lê a nota do usuário e verifica se ela é maior ou igual a 60. Se for, exibe a mensagem de aprovação. Caso contrário, exibe a mensagem de reprovação.

Exercício 36: Comparação de Três Números

Enunciado: Desenvolva um programa que leia três números inteiros e exiba o maior deles. Caso dois ou mais números sejam iguais, exiba uma mensagem indicando que há números iguais.

Código de solução:

```python
# Programa para encontrar o maior número ou verificar números iguais

# Solicita ao usuário três números inteiros
num1 = int(input("Digite o primeiro número inteiro: "))
num2 = int(input("Digite o segundo número inteiro: "))
num3 = int(input("Digite o terceiro número inteiro: "))

# Verifica se todos os números são iguais
if num1 == num2 == num3:
    print("Todos os números são iguais.")
# Verifica o maior número entre os três
elif num1 >= num2 and num1 >= num3:
    print(f"O maior número é: {num1}")
elif num2 >= num1 and num2 >= num3:
    print(f"O maior número é: {num2}")
else:
    print(f"O maior número é: {num3}")
```

Explicação: O programa solicita três números inteiros ao usuário. Ele verifica se todos os números são iguais e, caso não sejam, exibe o maior número entre eles..

Exercício 37: Avaliação de Temperatura

Enunciado: Escreva um programa que leia a temperatura atual em graus Celsius e exiba uma mensagem dizendo se o clima está "Frio" (abaixo de 15ºC), "Agradável" (entre 15ºC e 30ºC) ou "Quente" (acima de 30ºC).

Código de solução:

```
# Programa para avaliar a temperatura e classificar o clima

# Solicita a temperatura em graus Celsius
temperatura = float(input("Digite a temperatura em graus Celsius: "))

# Classifica o clima com base na temperatura
if temperatura < 15:
    print("Clima está Frio.")
elif temperatura <= 30:
    print("Clima está Agradável.")
else:
    print("Clima está Quente.")
```

Explicação: O programa lê a temperatura fornecida pelo usuário e classifica o clima como "Frio", "Agradável" ou "Quente", com base nas faixas de temperatura definidas.

Exercício 38: Verificação de Nota Escolar

Enunciado: Desenvolva um programa que leia uma nota escolar (de 0 a 10) e classifique-a como "Insuficiente" (menor que 5), "Suficiente" (entre 5 e 7) ou "Bom" (maior que 7).

Código de solução:

```
# Programa para classificar uma nota escolar

# Solicita a nota escolar
nota_escolar = float(input("Digite sua nota escolar (0 a 10): "))

# Classifica a nota escolar
if nota_escolar < 5:
    print("Classificação: Insuficiente.")
elif nota_escolar <= 7:
    print("Classificação: Suficiente.")
else:
    print("Classificação: Bom.")
```

Explicação: O programa lê a nota escolar inserida pelo usuário e classifica o desempenho como "Insuficiente", "Suficiente" ou "Bom", conforme as faixas de nota definidas.

Exercício 39: Comparação de Idades

Enunciado: Escreva um programa que leia as idades de duas pessoas e exiba quem é mais velho. Caso as idades sejam iguais, exiba uma mensagem informando que as duas pessoas têm a mesma idade.

Código de solução:

```
# Programa para comparar as idades de duas pessoas

# Solicita as idades das duas pessoas
idade1 = int(input("Digite a idade da primeira pessoa: "))
idade2 = int(input("Digite a idade da segunda pessoa: "))

# Compara as idades e exibe a mensagem apropriada
if idade1 > idade2:
    print("A primeira pessoa é mais velha.")
elif idade1 < idade2:
    print("A segunda pessoa é mais velha.")
else:
    print("Ambas têm a mesma idade.")
```

Explicação: O programa solicita as idades de duas pessoas, compara-as e exibe qual delas é mais velha ou se ambas têm a mesma idade.

Exercício 40: Avaliação de Velocidade

Enunciado: Crie um programa que leia a velocidade de um veículo e exiba uma mensagem dizendo se o veículo está dentro do limite de velocidade (até 60 km/h), acima do limite (entre 61 km/h e 80 km/h) ou muito acima do limite (acima de 80 km/h).

Código de solução:

```
# Programa para avaliar a velocidade de um veículo

# Solicita a velocidade do veículo
velocidade = int(input("Digite a velocidade do veículo (km/h): "))

# Avalia a velocidade e exibe a mensagem apropriada
```

```
if velocidade <= 60:
    print("Veículo está dentro do limite de velocidade.")
elif velocidade <= 80:
    print("Veículo está acima do limite de velocidade.")
else:
    print("Veículo está muito acima do limite de velocidade.")
```

Explicação: O programa lê a velocidade do veículo e exibe se ele está dentro do limite de 60 km/h, acima do limite (entre 61 km/h e 80 km/h) ou muito acima do limite (acima de 80 km/h).

Switch Case

Exercício 41: Dia da Semana

Enunciado: Escreva um programa que leia um número inteiro de 1 a 7 e exiba o nome do dia da semana correspondente (1 para domingo, 2 para segunda-feira, etc.).

Código de solução:

```
# Programa para exibir o dia da semana correspondente a um número de 1 a
7

# Solicita ao usuário um número de 1 a 7
dia_semana = int(input("Digite um número de 1 a 7 para o dia da semana:
"))

# Verifica o número e exibe o dia correspondente
if dia_semana == 1:
    print("Domingo")
elif dia_semana == 2:
    print("Segunda-feira")
elif dia_semana == 3:
    print("Terça-feira")
elif dia_semana == 4:
    print("Quarta-feira")
elif dia_semana == 5:
    print("Quinta-feira")
elif dia_semana == 6:
    print("Sexta-feira")
elif dia_semana == 7:
    print("Sábado")
```

```
else:
    print("Número inválido! Digite um número de 1 a 7.")
```

Explicação: O programa lê um número inteiro entre 1 e 7 e usa uma estrutura condicional if-elif-else para exibir o nome do dia da semana correspondente. Se o número estiver fora do intervalo de 1 a 7, uma mensagem de erro é exibida.

Exercício 42: Classificação de Nota

Enunciado: Crie um programa que leia uma nota de 0 a 10 e classifique a nota de acordo com as seguintes categorias:

- 10: Excelente
- 8 e 9: Muito bom
- 6 e 7: Bom
- 5: Regular
- 0 a 4: Insuficiente

Código de solução:

```
# Programa para classificar uma nota de 0 a 10

# Solicita ao usuário uma nota de 0 a 10
nota = int(input("Digite uma nota de 0 a 10: "))

# Classifica a nota de acordo com as categorias
if nota == 10:
    print("Excelente")
elif nota == 9 or nota == 8:
    print("Muito bom")
elif nota == 7 or nota == 6:
    print("Bom")
elif nota == 5:
    print("Regular")
elif 0 <= nota <= 4:
    print("Insuficiente")
else:
    print("Nota inválida.")
```

Explicação: O programa solicita uma nota ao usuário e a classifica conforme as categorias definidas. Se a nota estiver fora do intervalo de 0 a 10, uma mensagem de erro será exibida. A estrutura condicional if-elif-else é usada para fazer as verificações.

Exercício 43: Operações Matemáticas

Enunciado: Desenvolva um programa que leia dois números e um operador (+, -, *, /) e realize a operação correspondente. Exiba o resultado no console.

Código de solução:

```
# Programa para realizar operações matemáticas com dois números

# Solicita ao usuário dois números
numero1 = float(input("Digite o primeiro número: "))
numero2 = float(input("Digite o segundo número: "))

# Solicita ao usuário o operador
operador = input("Digite o operador (+, -, *, /): ")

# Realiza a operação correspondente e exibe o resultado
if operador == '+':
    print(f"Resultado: {numero1 + numero2}")
elif operador == '-':
    print(f"Resultado: {numero1 - numero2}")
elif operador == '*':
    print(f"Resultado: {numero1 * numero2}")
elif operador == '/':
    if numero2 != 0:
        print(f"Resultado: {numero1 / numero2}")
    else:
        print("Divisão por zero não é permitida.")
else:
    print("Operador inválido.")
```

Explicação: O programa solicita dois números e um operador ao usuário. Ele usa estruturas condicionais if-elif-else para verificar o operador e realizar a operação correspondente. No caso de divisão, há uma verificação adicional para evitar a divisão por zero. Caso o operador digitado não seja válido, uma mensagem de erro é exibida.

Exercício 44: Estação do Ano

Enunciado: Escreva um programa que leia um número de 1 a 4 e exiba o nome da estação do ano correspondente:

- 1: Verão

- 2: Outono
- 3: Inverno
- 4: Primavera

Código de solução:

```
# Programa para exibir a estação do ano com base em um número de 1 a 4

# Solicita ao usuário um número de 1 a 4
estacao = int(input("Digite um número de 1 a 4 para a estação do ano:
"))

# Verifica e exibe a estação correspondente
if estacao == 1:
    print("Verão")
elif estacao == 2:
    print("Outono")
elif estacao == 3:
    print("Inverno")
elif estacao == 4:
    print("Primavera")
else:
    print("Número inválido. Digite um número de 1 a 4.")
```

Explicação: O programa lê um número de 1 a 4 e exibe a estação correspondente, usando a estrutura switch case. Números fora do intervalo resultam em uma mensagem de erro.

Exercício 45: Classificação de Idade

Enunciado: Crie um programa que leia a idade de uma pessoa e a classifique em uma das seguintes categorias:

0 a 12 anos: Criança
13 a 17 anos: Adolescente
18 a 59 anos: Adulto
60 anos ou mais: Idoso
Utilize a estrutura if-elif-else para definir as condições de classificação.

Código de solução:

```
# Solicita ao usuário que insira a idade
```

```
idade = int(input("Digite a idade: "))

# Classifica a idade nas categorias apropriadas
if 0 <= idade <= 12:
    print("Criança")
elif 13 <= idade <= 17:
    print("Adolescente")
elif 18 <= idade <= 59:
    print("Adulto")
else:
    print("Idoso")
```

Explicação:
O programa solicita ao usuário que digite a idade.
A estrutura if-elif-else é usada para verificar em qual intervalo de idade a entrada se encaixa.
Dependendo do intervalo, ele exibe a classificação correspondente:

- 0 a 12: Criança
- 13 a 17: Adolescente
- 18 a 59: Adulto
- 60 ou mais: Idoso

Essa abordagem em Python é mais simples e direta do que tentar usar algo semelhante a switch-case, que não lida bem com intervalos de valores.

Laços de Repetição

Exercício 46: Contagem Crescente

Enunciado: Escreva um programa que exiba os números de 1 a 10, em ordem crescente, usando um laço for.

Código de solução:

```
# Laço for que exibe os números de 1 a 10 em ordem crescente
for i in range(1, 11):
    print(i)
```

Explicação: O laço for utiliza a função range(1, 11), que gera uma sequência de números de 1 a 10 (o valor final 11 não é incluído).
A cada iteração, o valor de i é exibido com a função print(), mostrando os números de 1 a 10 em ordem crescente.

Exercício 47: Contagem Decrescente

Enunciado: Crie um programa que exiba os números de 10 a 1, em ordem decrescente, usando um laço for.

Código de solução:

```
# Laço for que exibe os números de 10 a 1 em ordem decrescente
for i in range(10, 0, -1):
    print(i)
```

Explicação: O laço for usa a função range(10, 0, -1), que gera uma sequência de números de 10 a 1.
O primeiro argumento (10) é o valor inicial.
O segundo argumento (0) indica que o laço vai até 1, pois o limite superior não é incluído.
O terceiro argumento (-1) indica que a contagem será decrescente, decrementando 1 a cada iteração.
A cada passo, o valor de i é exibido com a função print().

Exercício 48: Números Pares

Enunciado: Desenvolva um programa que exiba todos os números pares de 1 a 20.

Código de solução:

```
# Laço for que exibe todos os números pares de 1 a 20
for i in range(2, 21, 2):
    print(i)
```

Explicação: O laço for utiliza a função range(2, 21, 2), que gera uma sequência de números de 2 a 20, incrementando de 2 em 2.
O primeiro argumento (2) é o valor inicial.
O segundo argumento (21) indica que o laço vai até 20 (o limite superior não é incluído).
O terceiro argumento (2) é o passo da contagem, garantindo que apenas números pares sejam exibidos.
A função print() exibe cada número par no console.

Exercício 49: Soma de Números

Enunciado: Escreva um programa que calcule a soma de todos os números de 1 a 100.

Código de solução:

```
# Calcula a soma de todos os números de 1 a 100
soma = 0
for i in range(1, 101):
    soma += i

print(f"Soma de 1 a 100: {soma}")
```

Explicação: A variável soma é inicializada com o valor 0.
O laço for percorre os números de 1 a 100 usando a função range(1, 101).
Em cada iteração, o valor de i é somado à variável soma.
Ao final do laço, o valor total da soma é exibido com a função print().

Exercício 50: Tabuada

Enunciado: Crie um programa que exiba a tabuada de um número fornecido pelo usuário, de 1 a 10.

Código de solução:

```
# Programa que exibe a tabuada de um número fornecido pelo usuário, de 1 a 10
numero = int(input("Digite um número para a tabuada: "))

for i in range(1, 11):
    print(f"{numero} x {i} = {numero * i}")
```

Explicação: O programa solicita ao usuário que insira um número com a função input(), e o valor digitado é convertido para um inteiro usando int().
O laço for percorre os valores de 1 a 10 com range(1, 11).
A cada iteração, é exibida a multiplicação do número fornecido pelo usuário com o valor de i, usando a função print() com f-string para formatar a saída.

Exercício 51: Fatorial

Enunciado: Desenvolva um programa que calcule o fatorial de um número inteiro fornecido pelo usuário.

Código de solução:

```
# Programa para calcular o fatorial de um número inteiro fornecido pelo usuário

# Solicita ao usuário um número
```

```
num = int(input("Digite um número para calcular o fatorial: "))
fatorial = 1

# Calcula o fatorial usando um laço for
for i in range(1, num + 1):
    fatorial *= i

# Exibe o resultado
print(f"Fatorial de {num} é: {fatorial}")
```

Explicação:O programa lê um número inteiro fornecido pelo usuário e utiliza um laço for para multiplicar os números de 1 até o número fornecido, acumulando o resultado na variável fatorial. O resultado final é exibido ao usuário.

Exercício 52: Números Ímpares

Enunciado: Escreva um programa que exiba todos os números ímpares entre 1 e 50.

Código de solução:

```
# Programa para exibir todos os números ímpares entre 1 e 50

for i in range(1, 51, 2):
    print(i)
```

Explicação: O laço for em Python começa no número 1 e vai até 50 (incluindo 50), com um incremento de 2 em 2, garantindo que somente os números ímpares sejam exibidos.

Exercício 53: Média de Números

Enunciado: Crie um programa que leia 5 números inteiros do usuário e calcule a média deles.

Código de solução:

```
# Programa para ler 5 números inteiros e calcular a média

soma = 0

for i in range(1, 6):
    numero = int(input(f"Digite o número {i}: "))
    soma += numero
```

```
media = soma / 5
print(f"A média é: {media}")
```

Explicação: O programa utiliza um laço for que vai de 1 a 5. A cada iteração, o usuário insere um número que é somado à variável soma. Após o término do laço, a média é calculada dividindo a soma total por 5 e exibida no console.

Exercício 54: Contagem de Múltiplos de 3

Enunciado: Desenvolva um programa que conte quantos números entre 1 e 100 são múltiplos de 3.

Código de solução:

```
# Programa para contar quantos números entre 1 e 100 são múltiplos de 3

contagem = 0

for i in range(1, 101):
    if i % 3 == 0:
        contagem += 1

print(f"Quantidade de múltiplos de 3 entre 1 e 100: {contagem}")
```

Explicação: O programa utiliza um laço for que percorre os números de 1 a 100. Em cada iteração, verifica se o número atual é múltiplo de 3 usando a operação de módulo (%). Se for, incrementa a variável contagem. No final, exibe a quantidade total de múltiplos de 3 encontrados.

Exercício 55: Sequência de Fibonacci

Enunciado: Escreva um programa que exiba os primeiros 10 termos da sequência de Fibonacci.

Código de solução:

```
# Programa para exibir os primeiros 10 termos da sequência de Fibonacci

termo1, termo2 = 0, 1

print(f"{termo1} {termo2}", end=" ")
```

```
for i in range(3, 11):
    proximo_termo = termo1 + termo2
    print(proximo_termo, end=" ")
    termo1, termo2 = termo2, proximo_termo
```

Explicação: O programa exibe os primeiros 10 termos da sequência de Fibonacci. Ele começa com os valores iniciais 0 e 1, imprimindo-os. Em seguida, utiliza um laço for que calcula o próximo termo somando os dois termos anteriores. Após imprimir o próximo termo, atualiza os valores de termo1 e termo2.

Exercício 56: Produto de Números

Enunciado: Crie um programa que calcule o produto dos números inteiros de 1 a 10.

Código de solução:

```
# Programa para calcular o produto dos números inteiros de 1 a 10

produto = 1

for i in range(1, 11):
    produto *= i

print(f"O produto dos números de 1 a 10 é: {produto}")
```

Explicação: O programa usa um laço for que percorre os números de 1 a 10 e multiplica cada número pelo valor atual da variável produto. O resultado final é exibido ao término do laço.

Exercício 57: Soma de Pares e Ímpares Separadamente

Enunciado: Escreva um programa que some todos os números pares de 1 a 100 e, separadamente, todos os números ímpares de 1 a 100.

Código de solução:

```
# Programa que soma os números pares e ímpares de 1 a 100

soma_pares = 0
soma_impares = 0
```

```
for i in range(1, 101):
    if i % 2 == 0:
        soma_pares += i
    else:
        soma_impares += i

print(f"Soma dos números pares: {soma_pares}")
print(f"Soma dos números ímpares: {soma_impares}")
```

Explicação: O programa percorre os números de 1 a 100. Se o número for par, ele é adicionado à variável soma_pares. Caso contrário, ele é adicionado à variável soma_impares. As somas são exibidas ao final.

Exercício 58: Validação de Entrada (while)

Enunciado: Crie um programa que leia um número inteiro entre 1 e 10. Caso o valor seja inválido, continue pedindo a entrada até que um número válido seja fornecido.

Código de solução:

```
# Programa que solicita um número entre 1 e 10 até que o usuário forneça um número válido

numero = int(input("Digite um número entre 1 e 10: "))

while numero < 1 or numero > 10:
    print("Número inválido. Tente novamente.")
    numero = int(input("Digite um número entre 1 e 10: "))

print(f"Número válido: {numero}")
```

Explicação: O programa pede um número inteiro entre 1 e 10. Se o número for inválido (fora desse intervalo), ele continua pedindo uma nova entrada até que um número válido seja fornecido. Quando o número válido é informado, ele é exibido.

Exercício 59: Soma de Números (while)

Enunciado: Desenvolva um programa que leia números inteiros do usuário e exiba a soma acumulada. O programa deve terminar quando o usuário digitar o número zero.

Código de solução:

```
# Programa que soma números fornecidos pelo usuário até o número zero
ser digitado

soma_acumulada = 0
valor_digitado = int(input("Digite números para somar. Digite 0 para
parar.\nDigite um número: "))

while valor_digitado != 0:
    soma_acumulada += valor_digitado
    valor_digitado = int(input("Digite outro número: "))

print(f"Soma total: {soma_acumulada}")
```

Explicação: O programa solicita números ao usuário e os soma até que o número zero seja digitado. Quando o zero é inserido, o laço é encerrado e a soma total acumulada é exibida.

Exercício 60: Contagem de Números Positivos (while)

Enunciado: Escreva um programa que leia números inteiros e exiba quantos desses números são positivos. O programa deve parar quando o usuário digitar um número negativo.

Código de solução:

```
# Programa que conta quantos números positivos são inseridos pelo
usuário

contagem_positivos = 0
numero = int(input("Digite um número (negativo para parar): "))

while numero >= 0:
    if numero > 0:
        contagem_positivos += 1
    numero = int(input("Digite outro número (negativo para parar): "))

print(f"Quantidade de números positivos: {contagem_positivos}")
```

Explicação: O programa lê números inteiros fornecidos pelo usuário até que um número negativo seja inserido. Cada vez que um número positivo é digitado, a contagem de positivos é incrementada. Quando o número negativo é inserido, o programa termina e exibe o total de números positivos.

Exercício 61: Raiz Quadrada Aproximada (while)

Enunciado: Crie um programa que leia um número inteiro positivo e encontre a raiz quadrada aproximada desse número. Continue a tentativa até encontrar a aproximação correta.

Código de solução:

```
# Programa que encontra a raiz quadrada aproximada de um número inteiro positivo

numero = int(input("Digite um número inteiro positivo: "))

raiz_aprox = 0
while raiz_aprox * raiz_aprox < numero:
    raiz_aprox += 1

if raiz_aprox * raiz_aprox == numero:
    print(f"Raiz quadrada exata de {numero} é: {raiz_aprox}")
else:
    print(f"Raiz quadrada aproximada de {numero} é: {raiz_aprox}")
```

Explicação: O programa lê um número inteiro positivo e utiliza um laço while para incrementar a variável raiz_aprox até que o quadrado dessa variável seja igual ou superior ao número fornecido. Se a raiz quadrada for exata, ela é exibida; caso contrário, o valor obtido será uma aproximação.

Exercício 62: Multiplicação por Acumulação (while)

Enunciado: Desenvolva um programa que leia um número e multiplique esse número por 2 repetidamente até o valor exceder 1000.

Código de solução:

```
# Programa que multiplica um número por 2 repetidamente até o valor exceder 1000

numero = int(input("Digite um número: "))

while numero <= 1000:
    numero *= 2
    print(f"Valor após multiplicação: {numero}")
```

```
print(f"Valor final após multiplicação acumulada: {numero}")
```

Explicação: O programa lê um número e usa um laço while para multiplicá-lo por 2 a cada iteração. O processo continua até que o número ultrapasse 1000, exibindo o valor após cada multiplicação. O valor final é exibido ao final do processo.

Exercício 63: Senha Correta (do-while)

Enunciado: Escreva um programa que peça ao usuário para digitar uma senha. Continue pedindo a senha até que a senha correta seja digitada.

Código de solução:

```
# Programa que pede ao usuário para digitar uma senha até que a correta seja fornecida

senha_correta = "1234"
senha_digitada = ""

while senha_digitada != senha_correta:
    senha_digitada = input("Digite a senha: ")
    if senha_digitada != senha_correta:
        print("Senha incorreta. Tente novamente.")

print("Senha correta! Acesso concedido.")
```

Explicação: O programa usa um laço while para continuar pedindo a senha ao usuário até que a senha correta seja fornecida. Caso a senha esteja incorreta, o programa exibe uma mensagem e solicita novamente a entrada.

Exercício 64: Menu de Opções (do-while)

Enunciado: Crie um programa que exiba um menu de opções e permita ao usuário escolher uma ação (como somar dois números, subtrair, etc.). O menu deve continuar sendo exibido até o usuário escolher a opção de sair.

Código de solução:

```
# Programa com um menu de opções que permite realizar operações matemáticas

def menu():
```

```
    while True:
        print("Menu de Opções:")
        print("1. Somar dois números")
        print("2. Subtrair dois números")
        print("3. Multiplicar dois números")
        print("4. Dividir dois números")
        print("5. Sair")
        opcao = int(input("Escolha uma opção: "))

        if opcao == 1 or opcao == 2 or opcao == 3 or opcao == 4:
            num1 = float(input("Digite o primeiro número: "))
            num2 = float(input("Digite o segundo número: "))

            if opcao == 1:
                print(f"Resultado: {num1 + num2}")
            elif opcao == 2:
                print(f"Resultado: {num1 - num2}")
            elif opcao == 3:
                print(f"Resultado: {num1 * num2}")
            elif opcao == 4:
                if num2 != 0:
                    print(f"Resultado: {num1 / num2}")
                else:
                    print("Erro: Divisão por zero não é permitida.")
        elif opcao == 5:
            print("Programa encerrado.")
            break
        else:
            print("Opção inválida. Tente novamente.")

# Chamada da função
menu()
```

Explicação: O programa apresenta um menu com opções de operações matemáticas. Ele continua exibindo o menu até que o usuário escolha a opção de sair (opção 5). O programa executa a operação correspondente à opção escolhida e exibe o resultado. Caso o usuário forneça uma opção inválida, o programa informa e pede novamente.

Conclusão do Capítulo 4: Controle de Fluxo

Neste capítulo, você explorou as principais ferramentas de controle de fluxo em Python, essenciais para criar programas dinâmicos e adaptáveis. As Estruturas Condicionais (if, elif, else) permitiram que você tomasse decisões com base em condições específicas, enquanto o comando match-case trouxe uma forma mais clara e direta de lidar com múltiplas opções possíveis.

Nos Laços de Repetição (for, while), você praticou a execução repetida de blocos de código, seja para iterar sobre números, validar entradas ou realizar cálculos complexos. Cada um desses laços tem seu próprio uso ideal, e agora você entende quando aplicar cada um deles.

Capítulo 5: Arrays e Coleções

Neste capítulo, você explorará o mundo das Listas e Coleções em Python, ferramentas essenciais para armazenar e manipular grandes volumes de dados de forma eficiente. Iniciaremos com as Listas (equivalentes aos Arrays unidimensionais), que permitem armazenar e acessar múltiplos valores de um mesmo tipo de dado de forma estruturada. A seguir, veremos as Listas de Listas, que funcionam como Arrays Multidimensionais (2D), permitindo organizar dados em matrizes e tabelas, aumentando as possibilidades de manipulação.

Além disso, aprenderemos a realizar operações essenciais de manipulação de Listas, como busca de elementos, ordenação de valores e outras técnicas amplamente usadas em desenvolvimento de software para organizar e processar dados de forma otimizada. Ao final deste capítulo, você será capaz de trabalhar com listas de diferentes dimensões e manipular esses dados de forma eficaz em diversos cenários práticos.

Arrays Unidimensionais

Exercício 65: Criação e Inicialização de um Array

Enunciado: Crie um programa que declare um array de 5 números inteiros. Atribua valores a esse array e, em seguida, exiba os valores no console.

Código de solução:

```
# Criação e inicialização do array
numeros = [10, 20, 30, 40, 50]

# Exibição dos valores do array
for i in range(len(numeros)):
    print(f"Elemento {i}: {numeros[i]}")
```

Explicação: O programa cria uma lista com 5 números inteiros e inicializa seus valores diretamente. Um laço for é usado para percorrer e exibir cada valor da lista.

Exercício 66: Soma de Elementos de um Array

Enunciado: Desenvolva um programa que leia 5 números inteiros do usuário, armazene-os em um array e calcule a soma de todos os elementos.

Código de solução:

```
# Criação do array e inicialização da variável de soma
numeros = []
soma = 0

# Leitura dos 5 números inteiros
for i in range(5):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)
    soma += numero

# Exibição da soma dos elementos
print(f"A soma dos elementos do array é: {soma}")
```

Explicação: O programa solicita que o usuário insira 5 números inteiros, que são armazenados em uma lista. Em seguida, a soma de todos os elementos da lista é calculada e exibida.

Exercício 67: Média de Valores

Enunciado: Escreva um programa que leia 10 números inteiros e calcule a média dos valores inseridos, utilizando um array para armazenar os números.

Código de solução:

```
# Criação do array e inicialização da variável de soma
numeros = []
soma = 0
```

```
# Leitura dos 10 números inteiros
for i in range(10):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)
    soma += numero

# Cálculo da média
media = soma / len(numeros)
print(f"A média dos valores é: {media}")
```

Explicação: O programa solicita que o usuário insira 10 números inteiros, que são armazenados em uma lista. Em seguida, calcula a soma dos números e, por fim, divide pela quantidade de elementos para obter a média, exibindo o resultado.

Exercício 68: Valores Pares em um Array

Enunciado: Crie um programa que leia 8 números inteiros e exiba todos os valores pares armazenados no array.

Código de solução:

```
# Criação do array de 8 números inteiros
numeros = []

# Leitura dos 8 números inteiros
for i in range(8):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)

# Exibição dos números pares
print("Números pares no array:")
for numero in numeros:
    if numero % 2 == 0:
        print(numero)
```

Explicação: O programa solicita ao usuário a inserção de 8 números inteiros, os armazena em uma lista e, em seguida, percorre a lista exibindo apenas os números pares (aqueles divisíveis por 2).

Exercício 69: Menor e Maior Valor

Enunciado: Desenvolva um programa que leia 10 números inteiros e armazene-os em um array. Encontre e exiba o menor e o maior valor presentes no array.

Código de solução:

```
# Criação do array de 10 números inteiros
numeros = []

# Leitura dos 10 números inteiros
for i in range(10):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)

# Encontrar o maior e o menor valor
maior = menor = numeros[0]

for numero in numeros:
    if numero > maior:
        maior = numero
    if numero < menor:
        menor = numero

# Exibir os resultados
print(f"Maior valor: {maior}")
print(f"Menor valor: {menor}")
```

Explicação: O programa solicita ao usuário a inserção de 10 números inteiros, armazena-os em uma lista e, em seguida, percorre a lista para encontrar o maior e o menor valor. O resultado é exibido ao final.

Exercício 70: Contagem de Valores Positivos e Negativos

Enunciado: Escreva um programa que leia 15 números inteiros e, em seguida, exiba quantos desses números são positivos e quantos são negativos.

Código de solução:

```
# Criação da lista para armazenar os 15 números inteiros
numeros = []
positivos = negativos = 0

# Leitura dos 15 números inteiros
for i in range(15):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)

# Contagem dos números positivos e negativos
for numero in numeros:
    if numero > 0:
        positivos += 1
    elif numero < 0:
        negativos += 1

# Exibir os resultados
print(f"Quantidade de números positivos: {positivos}")
print(f"Quantidade de números negativos: {negativos}")
```

Explicação: O programa solicita ao usuário a inserção de 15 números inteiros, conta quantos são positivos e quantos são negativos, e exibe o resultado.

Exercício 71: Inversão de Array

Enunciado: Crie um programa que leia 6 números inteiros e armazene-os em um array. Depois, exiba os valores do array na ordem inversa.

Código de solução:

```
# Criação da lista para armazenar os 6 números inteiros
numeros = []

# Leitura dos 6 números inteiros
for i in range(6):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)

# Exibição dos números em ordem inversa
print("Números em ordem inversa:")
for i in range(len(numeros) - 1, -1, -1):
    print(numeros[i])
```

Explicação: O programa solicita ao usuário a inserção de 6 números inteiros, os armazena em uma lista e, em seguida, exibe os números na ordem inversa, começando do último elemento.

Exercício 72: Contagem de Ocorrências de um Número

Enunciado: Desenvolva um programa que leia 10 números inteiros e armazene-os em um array. O programa deve pedir ao usuário para inserir um número extra e contar quantas vezes esse número aparece no array.

Código de solução:

```
# Criação da lista para armazenar os 10 números inteiros
numeros = []

# Leitura dos 10 números inteiros
for i in range(10):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)

# Leitura do número para contar as ocorrências
numero_procurado = int(input("Digite um número para contar as
ocorrências: "))

# Contagem das ocorrências
ocorrencias = numeros.count(numero_procurado)
```

```
# Exibição do resultado
print(f"O número {numero_procurado} aparece {ocorrencias} vezes no
array.")
```

Explicação: O programa lê 10 números inteiros e os armazena em uma lista. Em seguida, pede ao usuário para inserir um número adicional e conta quantas vezes esse número aparece na lista, utilizando o método count() para facilitar a contagem das ocorrências.

Exercício 73: Duplicação de Valores

Enunciado: Escreva um programa que crie um array de 5 números inteiros e multiplique todos os seus valores por 2, exibindo o novo array no console.

Código de solução:

```
# Criação da lista para armazenar os 5 números inteiros
numeros = []

# Leitura dos 5 números inteiros
for i in range(5):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)

# Multiplicação dos valores por 2 e exibição do novo array
print("Valores duplicados:")
for i in range(len(numeros)):
    numeros[i] *= 2
    print(numeros[i])
```

Explicação: O programa lê 5 números inteiros, multiplica cada um deles por 2 e, em seguida, exibe os novos valores na tela.

Exercício 74: Elementos em Posições Ímpares

Enunciado: Crie um programa que leia 10 números inteiros e exiba apenas os valores que estão em posições ímpares no array.

Código de solução:

```
# Criação da lista para armazenar os 10 números inteiros
numeros = []

# Leitura dos 10 números inteiros
for i in range(10):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)

# Exibição dos valores nas posições ímpares
print("Valores nas posições ímpares:")
for i in range(1, len(numeros), 2):
    print(numeros[i])
```

Explicação: O programa lê 10 números inteiros e exibe apenas os valores que estão nas posições ímpares do array (índices 1, 3, 5, etc.).

Exercício 75: Substituição de Valores em um Array

Enunciado: Desenvolva um programa que crie um array de 10 números inteiros. O programa deve pedir ao usuário que forneça dois números: um número para buscar no array e outro para substituir o número encontrado. Se o número for encontrado, ele deve ser substituído.

Código de solução:

```
# Criação da lista para armazenar os 10 números inteiros
numeros = []

# Leitura dos 10 números inteiros
for i in range(10):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)

# Leitura dos números a serem substituídos
numero_antigo = int(input("Digite o número a ser substituído: "))
numero_novo = int(input("Digite o novo número: "))
```

```
# Flag para verificar se o número foi encontrado
encontrado = False

# Substituição do número
for i in range(len(numeros)):
    if numeros[i] == numero_antigo:
        numeros[i] = numero_novo
        encontrado = True

# Exibição do resultado
if encontrado:
    print("O número foi substituído. Novo array:")
    for numero in numeros:
        print(numero)
else:
    print("Número não encontrado no array.")
```

Explicação: O programa solicita ao usuário que forneça um número para buscar e substituir no array. Se o número for encontrado, ele é substituído, e o array atualizado é exibido. Caso contrário, uma mensagem informa que o número não foi encontrado.

Exercício 76: Verificação de Elementos Repetidos

Enunciado: Escreva um programa que leia 10 números inteiros e verifique se algum valor é repetido no array. Se houver repetições, exiba uma mensagem informando.

Código de solução:

```
# Criação da lista para armazenar os 10 números inteiros
numeros = []

# Leitura dos 10 números inteiros
for i in range(10):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)

# Verificação de repetições
```

```
repetido = False
for i in range(len(numeros)):
    for j in range(i + 1, len(numeros)):
        if numeros[i] == numeros[j]:
            repetido = True
            break

# Exibição do resultado
if repetido:
    print("Há valores repetidos no array.")
else:
    print("Não há valores repetidos no array.")
```

Explicação: O programa lê 10 números inteiros e compara cada número com os outros para verificar se há repetições. Caso algum valor repetido seja encontrado, o programa exibe uma mensagem informando sobre isso. Se não houver repetições, ele informa que não há valores repetidos no array.

Exercício 77: Produto dos Elementos de um Array

Enunciado: Crie um programa que leia 6 números inteiros e calcule o produto de todos os valores do array.

Código de solução:

```
# Criação da lista para armazenar os 6 números inteiros
numeros = []
produto = 1

# Leitura dos 6 números inteiros
for i in range(6):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)
    produto *= numero

# Exibição do resultado
print(f"O produto dos elementos do array é: {produto}")
```

Explicação: O programa lê 6 números inteiros e calcula o produto de todos os valores armazenados na lista. O resultado é exibido após o cálculo.

Exercício 78: Comparação de Arrays

Enunciado: Desenvolva um programa que crie dois arrays de 5 números inteiros. O programa deve comparar os dois arrays e exibir quais posições possuem valores iguais.

Código de solução:

```
# Criação de dois arrays (listas) de 5 números inteiros
array1 = []
array2 = []

# Leitura dos elementos para o primeiro array
print("Preencha o primeiro array:")
for i in range(5):
    numero = int(input(f"Elemento {i + 1}: "))
    array1.append(numero)

# Leitura dos elementos para o segundo array
print("Preencha o segundo array:")
for i in range(5):
    numero = int(input(f"Elemento {i + 1}: "))
    array2.append(numero)

# Comparação dos elementos nas mesmas posições
print("Comparação de valores nas mesmas posições:")
for i in range(len(array1)):
    if array1[i] == array2[i]:
        print(f"Posição {i}: {array1[i]} = {array2[i]}")
```

Explicação: O programa lê dois arrays (listas) de 5 números inteiros e compara os elementos nas mesmas posições. Quando os valores nas posições correspondentes são iguais, o programa exibe essas informações.

Exercício 79: Verificação de Ordem Crescente

Enunciado: Escreva um programa que leia 8 números inteiros e verifique se os valores estão em ordem crescente. Exiba uma mensagem indicando se os números estão ou não em ordem.

Código de solução:

```
# Leitura dos 8 números inteiros
numeros = []
em_ordem = True

for i in range(8):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)

# Verificação se os números estão em ordem crescente
for i in range(len(numeros) - 1):
    if numeros[i] > numeros[i + 1]:
        em_ordem = False
        break

# Exibição da mensagem
if em_ordem:
    print("Os números estão em ordem crescente.")
else:
    print("Os números NÃO estão em ordem crescente.")
```

Explicação: O programa lê 8 números inteiros, armazena-os em uma lista e verifica se os números estão em ordem crescente. Se algum número for maior do que o próximo, ele exibe uma mensagem indicando que os números não estão em ordem crescente. Caso contrário, ele confirma que estão em ordem crescente.

Arrays Multidimensionais (2D) (10 exercícios)

Dica: Arrays multidimensionais podem ser criados utilizando a sintaxe: int[][] x = [2][2], sendo assim este array terá 2 linhas e duas colunas.

Exercício 80: Criação de uma Matriz

Enunciado: Crie um programa que declare uma matriz 3x3 e permita que o usuário insira valores inteiros para preencher essa matriz. Em seguida, exiba os valores da matriz no console.

Código de solução:

```
# Criando uma matriz 3x3
matriz = [[0 for _ in range(3)] for _ in range(3)]

# Preenchendo a matriz com valores fornecidos pelo usuário
for i in range(3):
    for j in range(3):
        matriz[i][j] = int(input(f"Digite o valor para a posição
[{i}][{j}]: "))

# Exibindo a matriz
print("\nMatriz 3x3:")
for linha in matriz:
    print(" ".join(map(str, linha)))
```

Explicação: Criação da matriz: A matriz é inicializada com valores 0 utilizando list comprehension.
Entrada de dados: Dois loops for são usados para iterar pelas linhas e colunas, solicitando ao usuário que insira os valores correspondentes.
Exibição da matriz: Cada linha da matriz é exibida usando join para formatar a saída em uma tabela legível.

Exercício 81: Soma de Elementos de uma Matriz

Enunciado: Desenvolva um programa que leia uma matriz 3x3 de inteiros e calcule a soma de todos os elementos da matriz.

Código de solução:

```
# Criando uma matriz 3x3 e inicializando a soma
matriz = [[0 for _ in range(3)] for _ in range(3)]
soma = 0

# Preenchendo a matriz e calculando a soma
for i in range(3):
    for j in range(3):
        matriz[i][j] = int(input(f"Digite o valor para a posição
[{i}][{j}]: "))
        soma += matriz[i][j]

# Exibindo a soma dos elementos
```

```
print(f"\nA soma de todos os elementos da matriz é: {soma}")
```

Explicação: Criação da matriz: A matriz 3x3 é criada usando list comprehension.
Entrada de dados: Dois loops for são usados para iterar pelas linhas e colunas, pedindo ao usuário para inserir os valores correspondentes.
Cálculo da soma: Cada valor inserido é adicionado à variável soma.
Exibição: A soma total dos elementos é exibida no final.

Exercício 82: Soma das Linhas e Colunas

Enunciado: Escreva um programa que leia uma matriz 3x3 e exiba a soma dos elementos de cada linha e de cada coluna.

Código de solução:

```
# Criando a matriz 3x3
matriz = [[0 for _ in range(3)] for _ in range(3)]
soma_linhas = [0 for _ in range(3)]
soma_colunas = [0 for _ in range(3)]

# Preenchendo a matriz e calculando as somas
for i in range(3):
    for j in range(3):
        matriz[i][j] = int(input(f"Digite o valor para a posição
[{i}][{j}]: "))
        soma_linhas[i] += matriz[i][j]
        soma_colunas[j] += matriz[i][j]

# Exibindo a soma das linhas
for i in range(3):
    print(f"Soma da linha {i}: {soma_linhas[i]}")

# Exibindo a soma das colunas
for j in range(3):
    print(f"Soma da coluna {j}: {soma_colunas[j]}")
```

Explicação: Criação da matriz e vetores de soma: A matriz 3x3 e os vetores soma_linhas e soma_colunas são inicializados com zeros.
Entrada de dados e cálculo das somas: Durante o preenchimento da matriz, as somas de cada linha e coluna são acumuladas diretamente.
Exibição das somas: O programa exibe separadamente a soma de cada linha e de cada coluna.

Exercício 83: Matriz Identidade

Enunciado: Crie um programa que gere e exiba uma matriz identidade 4x4 (valores 1 na diagonal principal e 0 nos outros elementos).

Código de solução:

```
# Criando a matriz 4x4 vazia
matriz = [[0 for _ in range(4)] for _ in range(4)]

# Gerando a matriz identidade
for i in range(4):
    matriz[i][i] = 1  # Elementos da diagonal principal recebem 1

# Exibindo a matriz identidade
print("Matriz identidade 4x4:")
for i in range(4):
    for j in range(4):
        print(matriz[i][j], end=" ")
    print()  # Quebra de linha após cada linha da matriz
```

Explicação: Criação da matriz: A matriz é inicializada como uma lista de listas com valores 0.
Geração da matriz identidade: Os elementos da diagonal principal (onde o índice da linha é igual ao da coluna) recebem o valor 1.
Exibição da matriz: A matriz é exibida no console, com uma quebra de linha ao final de cada linha.

Exercício 84: Busca em uma Matriz

Enunciado: Desenvolva um programa que permita ao usuário preencher uma matriz 4x4 com valores inteiros. O programa deve pedir ao usuário um número para buscar na matriz e informar em qual posição ele foi encontrado (linha e coluna).

Código de solução:

```
# Criando a matriz 4x4 vazia
matriz = [[0 for _ in range(4)] for _ in range(4)]

# Preenchendo a matriz com valores inteiros
for i in range(4):
```

```
    for j in range(4):
        matriz[i][j] = int(input(f"Digite o valor para a posição
[{i}][{j}]: "))

# Solicitando ao usuário um número para buscar na matriz
numero = int(input("Digite o número para buscar: "))
encontrado = False

# Buscando o número na matriz
for i in range(4):
    for j in range(4):
        if matriz[i][j] == numero:
            print(f"Número encontrado na posição [{i}][{j}]")
            encontrado = True

if not encontrado:
    print("Número não encontrado na matriz.")
```

Explicação: Criação da matriz: Inicializa-se uma matriz 4x4 com valores zero usando list comprehension.
Preenchimento da matriz: O programa solicita ao usuário que insira valores inteiros para cada posição da matriz.
Busca do número: Após o preenchimento, o programa solicita um número para buscar na matriz. Se o número for encontrado, exibe a posição (linha e coluna) correspondente.
Mensagem de resultado: Se o número não for encontrado, exibe uma mensagem informando que o número não está presente na matriz.

Exercício 85: Matriz Transposta

Enunciado: Escreva um programa que leia uma matriz 3x3 e exiba a sua matriz transposta (inversão das linhas com as colunas).

Código de solução:

```
# Criando matrizes 3x3 vazias
matriz = [[0 for _ in range(3)] for _ in range(3)]
transposta = [[0 for _ in range(3)] for _ in range(3)]

# Preenchendo a matriz e gerando a transposta simultaneamente
for i in range(3):
    for j in range(3):
        matriz[i][j] = int(input(f"Digite o valor para a posição
[{i}][{j}]: "))
```

```
        transposta[j][i] = matriz[i][j]

# Exibindo a matriz transposta
print("Matriz transposta 3x3:")
for i in range(3):
    for j in range(3):
        print(transposta[i][j], end=" ")
    print()
```

Explicação: Criação das matrizes: Inicializa-se duas matrizes 3x3: matriz (para armazenar os valores fornecidos pelo usuário) e transposta (para armazenar a matriz transposta). Preenchimento da matriz: Solicita-se ao usuário os valores de cada posição da matriz, preenchendo simultaneamente a matriz original e a transposta (invertendo os índices i e j). Exibição da matriz transposta: Percorre-se a matriz transposta e exibe seus elementos no formato correto.

Exercício 86: Diagonal Principal e Secundária

Enunciado: Crie um programa que leia uma matriz 4x4 e exiba os elementos da diagonal principal e da diagonal secundária.

Código de solução:

```
# Inicializa a matriz 4x4
matriz = [[0 for _ in range(4)] for _ in range(4)]

# Preenchendo a matriz
for i in range(4):
    for j in range(4):
        matriz[i][j] = int(input(f"Digite o valor para a posição
[{i}][{j}]: "))

# Exibindo a diagonal principal
print("Diagonal principal:")
for i in range(4):
    print(matriz[i][i])

# Exibindo a diagonal secundária
print("Diagonal secundária:")
for i in range(4):
    print(matriz[i][3 - i])
```

Explicação: Criação da matriz: Uma matriz 4x4 é criada e preenchida com valores inseridos pelo usuário.
Diagonal principal: A diagonal principal é composta pelos elementos onde a linha e a coluna possuem o mesmo índice (matriz[i][i]).
Diagonal secundária: A diagonal secundária é composta pelos elementos onde a linha e a coluna têm índices opostos, ou seja, a coluna é dada por 3 - i (a partir da última coluna indo até a primeira).

Exercício 87: Multiplicação de Matrizes

Enunciado: Desenvolva um programa que leia duas matrizes 2x2 e calcule o produto entre elas, exibindo o resultado.

Código de solução:

```
# Inicializa as matrizes 2x2
matrizA = [[0 for _ in range(2)] for _ in range(2)]
matrizB = [[0 for _ in range(2)] for _ in range(2)]
resultado = [[0 for _ in range(2)] for _ in range(2)]

# Preenchendo a matriz A
print("Preenchendo a matriz A:")
for i in range(2):
    for j in range(2):
        matrizA[i][j] = int(input(f"Digite o valor para a posição
[{i}][{j}] da matriz A: "))

# Preenchendo a matriz B
print("Preenchendo a matriz B:")
for i in range(2):
    for j in range(2):
        matrizB[i][j] = int(input(f"Digite o valor para a posição
[{i}][{j}] da matriz B: "))

# Multiplicação das matrizes A e B
for i in range(2):
    for j in range(2):
        resultado[i][j] = 0
        for k in range(2):
            resultado[i][j] += matrizA[i][k] * matrizB[k][j]

# Exibindo o resultado da multiplicação
print("Resultado da multiplicação das matrizes A e B:")
for i in range(2):
```

```
    for j in range(2):
        print(resultado[i][j], end=" ")
    print()
```

Explicação: Preenchimento das matrizes A e B: O programa solicita ao usuário que preencha as duas matrizes 2x2.
Multiplicação das matrizes: A multiplicação é feita utilizando a regra de multiplicação de matrizes, onde o elemento resultado[i][j] é a soma dos produtos dos elementos correspondentes das linhas da matriz A e das colunas da matriz B.
Exibição do resultado: O programa exibe a matriz resultante após a multiplicação.

Exercício 88: Soma de Duas Matrizes

Enunciado: Escreva um programa que leia duas matrizes 3x3 e calcule a soma entre elas, exibindo a matriz resultante.

Código de solução:

```
# Inicializa as matrizes 3x3
matrizA = [[0 for _ in range(3)] for _ in range(3)]
matrizB = [[0 for _ in range(3)] for _ in range(3)]
soma = [[0 for _ in range(3)] for _ in range(3)]

# Preenchendo a matriz A
print("Preenchendo a matriz A:")
for i in range(3):
    for j in range(3):
        matrizA[i][j] = int(input(f"Digite o valor para a posição
[{i}][{j}] da matriz A: "))

# Preenchendo a matriz B
print("Preenchendo a matriz B:")
for i in range(3):
    for j in range(3):
        matrizB[i][j] = int(input(f"Digite o valor para a posição
[{i}][{j}] da matriz B: "))

# Somando as duas matrizes
for i in range(3):
    for j in range(3):
        soma[i][j] = matrizA[i][j] + matrizB[i][j]

# Exibindo a matriz resultante
print("Soma das matrizes A e B:")
```

```
for i in range(3):
    for j in range(3):
        print(soma[i][j], end=" ")
    print()
```

Explicação: Leitura das Matrizes A e B: O programa pede ao usuário para preencher duas matrizes 3x3.
Soma das Matrizes: A soma é feita somando os elementos correspondentes das duas matrizes, ou seja, soma[i][j] = matrizA[i][j] + matrizB[i][j].
Exibição da Matriz Resultante: O programa imprime a matriz resultante da soma.

Exercício 89: Contagem de Elementos Pares

Enunciado: Crie um programa que leia uma matriz 5x5 e conte quantos números pares existem na matriz. Exiba o total de números pares encontrados.

Código de solução:

```
# Inicializa a matriz 5x5
matriz = [[0 for _ in range(5)] for _ in range(5)]
contagemPares = 0

# Preenchendo a matriz e contando os números pares
for i in range(5):
    for j in range(5):
        matriz[i][j] = int(input(f"Digite o valor para a posição
[{i}][{j}]: "))
        if matriz[i][j] % 2 == 0:
            contagemPares += 1

# Exibindo o número de números pares
print(f"Total de números pares na matriz: {contagemPares}")
```

Explicação: Leitura da Matriz: O programa preenche uma matriz 5x5 com valores fornecidos pelo usuário.
Contagem de Números Pares: O programa verifica se cada número é par (usando matriz[i][j] % 2 == 0). Se for, incrementa a variável contagemPares.
Exibição do Resultado: Após percorrer toda a matriz, o programa exibe o total de números pares encontrados.

Manipulação de Arrays (busca, ordenação, etc.)

Exercício 90: Busca Linear em um Array

Enunciado: Crie um programa que leia 10 números inteiros e um número adicional. O programa deve realizar uma busca linear no array para verificar se o número adicional está presente. Exiba a posição do número, se encontrado.

Código de solução:

```
# Inicializa o array de 10 números
numeros = []

# Preenchendo o array com 10 números inteiros
for i in range(10):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)

# Lendo o número adicional
numeroBuscado = int(input("Digite o número que deseja buscar: "))

# Realizando a busca linear
encontrado = False
for i in range(len(numeros)):
    if numeros[i] == numeroBuscado:
        print(f"Número encontrado na posição: {i}")
        encontrado = True
        break

if not encontrado:
    print("Número não encontrado.")
```

Explicação: Leitura dos Números: O programa lê 10 números inteiros e os armazena em uma lista numeros.
Busca Linear: O programa realiza uma busca linear para verificar se o número adicional está presente na lista. Caso o número seja encontrado, ele exibe a posição do número.
Resultado: Se o número for encontrado, o programa exibe sua posição (considerando que a contagem de índices começa em 0). Caso contrário, exibe "Número não encontrado."

Exercício 91: Busca Binária em um Array Ordenado

Enunciado: Desenvolva um programa que leia 10 números inteiros, os ordene em ordem crescente e, em seguida, utilize o método de busca binária para encontrar um número fornecido pelo usuário. Exiba a posição do número se ele for encontrado.

Código de solução:

```
def busca_binaria(arr, x):
    esquerda, direita = 0, len(arr) - 1
    while esquerda <= direita:
        meio = (esquerda + direita) // 2
        if arr[meio] == x:
            return meio  # Número encontrado, retorna a posição
        elif arr[meio] < x:
            esquerda = meio + 1  # Busca na metade direita
        else:
            direita = meio - 1  # Busca na metade esquerda
    return -1  # Número não encontrado

# Lendo os 10 números inteiros
numeros = []
for i in range(10):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)

# Ordenando o array em ordem crescente
numeros.sort()

# Lendo o número a ser buscado
numero_buscado = int(input("Digite o número que deseja buscar: "))

# Realizando a busca binária
posicao = busca_binaria(numeros, numero_buscado)

if posicao != -1:
    print(f"Número encontrado na posição: {posicao}")
else:
    print("Número não encontrado.")
```

Explicação: Leitura dos Números: O programa lê 10 números inteiros e os armazena na lista numeros.
Ordenação: A lista é ordenada em ordem crescente usando o método sort().
Busca Binária: A função busca_binaria é implementada para realizar a busca binária no array ordenado. Ela retorna a posição do número se encontrado, ou -1 se não encontrado.

Resultado: Se o número for encontrado, o programa exibe sua posição na lista ordenada. Caso contrário, ele informa que o número não foi encontrado.

Exercício 92: Ordenação de um Array com Bubble Sort

Enunciado: Implemente o algoritmo de ordenação Bubble Sort para ordenar um array de 10 números inteiros em ordem crescente.

Código de solução:

```
def busca_binaria(arr, x):
    esquerda, direita = 0, len(arr) - 1
    while esquerda <= direita:
        meio = (esquerda + direita) // 2
        if arr[meio] == x:
            return meio  # Número encontrado, retorna a posição
        elif arr[meio] < x:
            esquerda = meio + 1  # Busca na metade direita
        else:
            direita = meio - 1  # Busca na metade esquerda
    return -1  # Número não encontrado

# Lendo os 10 números inteiros
numeros = []
for i in range(10):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)

# Ordenando o array em ordem crescente
numeros.sort()

# Lendo o número a ser buscado
numero_buscado = int(input("Digite o número que deseja buscar: "))

# Realizando a busca binária
posicao = busca_binaria(numeros, numero_buscado)

if posicao != -1:
    print(f"Número encontrado na posição: {posicao}")
else:
    print("Número não encontrado.")
```

Explicação: Leitura dos Números: O programa lê 10 números inteiros e os armazena na lista numeros.
Ordenação: A lista é ordenada em ordem crescente usando o método sort().

Busca Binária: A função busca_binaria é implementada para realizar a busca binária no array ordenado. Ela retorna a posição do número se encontrado, ou -1 se não encontrado.
Resultado: Se o número for encontrado, o programa exibe sua posição na lista ordenada. Caso contrário, ele informa que o número não foi encontrado.

Exercício 93: Ordenação de um Array com Selection Sort

Enunciado: Escreva um programa que leia 10 números inteiros e os ordene utilizando o algoritmo Selection Sort. Exiba o array ordenado ao final.

Código de solução:

```
# Lendo o array de 10 números inteiros
numeros = []
for i in range(10):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)

# Algoritmo de ordenação Selection Sort
for i in range(len(numeros) - 1):
    min_index = i
    for j in range(i + 1, len(numeros)):
        if numeros[j] < numeros[min_index]:
            min_index = j

    # Troca o menor elemento encontrado com o elemento da posição i
    numeros[i], numeros[min_index] = numeros[min_index], numeros[i]

# Exibindo o array ordenado
print("Array ordenado:")
for numero in numeros:
    print(numero, end=" ")
```

Explicação: Leitura dos Números: O programa lê 10 números inteiros e os armazena na lista numeros.
Algoritmo Selection Sort: O algoritmo percorre o array e encontra o menor elemento não ordenado, trocando-o com o elemento da posição atual. Esse processo é repetido para cada posição do array, até que ele esteja completamente ordenado.
Exibição do Resultado: Após a ordenação, o programa imprime os números ordenados.

Exercício 94: Ordenação de um Array com Insertion Sort

Enunciado: Desenvolva um programa que leia 10 números inteiros e os ordene utilizando o algoritmo Insertion Sort. Exiba o array ordenado ao final.

Código de solução:

```
# Lendo o array de 10 números inteiros
numeros = []
for i in range(10):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)

# Algoritmo de ordenação Insertion Sort
for i in range(1, len(numeros)):
    chave = numeros[i]  # Número a ser inserido na posição correta
    j = i - 1

    # Desloca os elementos maiores que a chave uma posição à frente
    while j >= 0 and numeros[j] > chave:
        numeros[j + 1] = numeros[j]
        j -= 1

    # Insere a chave na posição correta
    numeros[j + 1] = chave

# Exibindo o array ordenado
print("Array ordenado:")
for numero in numeros:
    print(numero, end=" ")
```

Explicação: Leitura dos Números: O programa lê 10 números inteiros e os armazena na lista numeros.
Algoritmo Insertion Sort: O algoritmo começa no segundo elemento do array e o insere na posição correta, movendo os elementos maiores uma posição para a direita. Esse processo é repetido para cada elemento até que o array esteja ordenado.
Exibição do Resultado: Após a ordenação, o programa imprime os números ordenados.
O Insertion Sort é eficiente para listas pequenas ou quase ordenadas, pois, embora sua complexidade no pior caso seja $O(n^2)$, ele realiza poucas trocas em listas parcialmente ordenadas.

Exercício 95: Inversão de um Array

Enunciado: Crie um programa que leia 10 números inteiros, armazene-os em um array e inverta a ordem dos elementos no array. Exiba o array invertido.

Código de solução:

```
# Lendo o array de 10 números inteiros
numeros = []
for i in range(10):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)

# Invertendo o array
for i in range(len(numeros) // 2):
    # Troca de elementos
    temp = numeros[i]
    numeros[i] = numeros[len(numeros) - 1 - i]
    numeros[len(numeros) - 1 - i] = temp

# Exibindo o array invertido
print("Array invertido:")
for numero in numeros:
    print(numero, end=" ")
```

Explicação: Leitura dos Números: O programa lê 10 números inteiros e os armazena na lista numeros.
Inversão do Array: O algoritmo percorre a primeira metade da lista e troca cada elemento com o elemento correspondente da segunda metade, utilizando uma variável temporária temp.
Exibição do Resultado: Após inverter os elementos, o programa imprime o array invertido.

Exercício 96: Remoção de Elemento de um Array

Enunciado: Escreva um programa que leia 10 números inteiros e remova um número específico informado pelo usuário. Após a remoção, exiba o array resultante.

Código de solução:

```
# Lendo os 10 números inteiros
numeros = []
for i in range(10):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)

# Lendo o número a ser removido
numero_remover = int(input("Digite o número a ser removido: "))
encontrado = False
```

```
# Criando o novo array sem o número removido
novo_array = []

for numero in numeros:
    if numero == numero_remover and not encontrado:
        encontrado = True
    else:
        novo_array.append(numero)

# Exibindo o novo array
if encontrado:
    print("Array após remoção:")
    print(novo_array)
else:
    print("Número não encontrado no array.")
```

Explicação: Leitura dos Números: O programa lê 10 números inteiros e os armazena na lista numeros.
Leitura do Número para Remoção: O número a ser removido é informado pelo usuário.
Remoção: O programa cria uma nova lista, novo_array, onde adiciona todos os números, exceto o número que foi removido. A remoção ocorre apenas na primeira ocorrência do número.
Exibição do Resultado: O programa exibe o novo array resultante após a remoção. Se o número não for encontrado, uma mensagem informando isso será exibida.

Exercício 97: Inserção de Elemento em um Array Ordenado

Enunciado: Desenvolva um programa que leia 10 números inteiros, os ordene em ordem crescente e insira um novo número no array mantendo a ordem. Exiba o array resultante.

Código de solução:

```
# Lendo os 10 números inteiros
numeros = []
for i in range(10):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)

# Ordenando o array em ordem crescente
numeros.sort()

# Lendo o número a ser inserido
numero_inserir = int(input("Digite o número a ser inserido: "))
```

```
# Criando o novo array com espaço adicional
novo_array = []

# Inserindo o número no array ordenado
i = 0
while i < len(numeros) and numeros[i] < numero_inserir:
    novo_array.append(numeros[i])
    i += 1
novo_array.append(numero_inserir)  # Insere o novo número
while i < len(numeros):
    novo_array.append(numeros[i])
    i += 1

# Exibindo o novo array
print("Array após a inserção:")
print(novo_array)
```

Explicação: Leitura dos Números: O programa lê 10 números inteiros e os armazena na lista numeros.
Ordenação: Os números são ordenados em ordem crescente usando o método sort().
Inserção do Novo Número: O programa pede ao usuário um número para ser inserido. A inserção ocorre mantendo a ordem crescente, onde o número é colocado na posição correta do novo array.
Exibição do Resultado: O programa exibe o novo array após a inserção do número.

Exercício 98: Contagem de Elementos em um Intervalo

Enunciado: Crie um programa que leia 15 números inteiros e conte quantos números estão dentro de um intervalo fornecido pelo usuário. Exiba a quantidade.

Código de solução:

```
# Lendo os 15 números inteiros
numeros = []
for i in range(15):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)

# Lendo o intervalo
limite_inferior = int(input("Digite o limite inferior do intervalo: "))
limite_superior = int(input("Digite o limite superior do intervalo: "))

# Contando os números no intervalo
```

```
contagem = 0
for numero in numeros:
    if limite_inferior <= numero <= limite_superior:
        contagem += 1

# Exibindo a contagem
print("Quantidade de números no intervalo:", contagem)
```

Explicação: Leitura dos Números: O programa lê 15 números inteiros e os armazena na lista numeros. Leitura do Intervalo: O programa lê dois valores que definem o intervalo, limite_inferior e limite_superior. Contagem de Números no Intervalo: O programa percorre a lista de números e verifica se cada número está dentro do intervalo especificado. Se estiver, incrementa o contador contagem. Exibição da Contagem: O programa exibe o total de números que estão dentro do intervalo fornecido.

Exercício 99: Duplicação de Elementos de um Array

Enunciado: Escreva um programa que leia 10 números inteiros e duplique todos os valores do array. Exiba o array duplicado.

Código de solução:

```
# Lendo os 10 números inteiros
numeros = []
for i in range(10):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)

# Duplicando os valores
for i in range(len(numeros)):
    numeros[i] *= 2

# Exibindo o array duplicado
print("Array após a duplicação:")
for numero in numeros:
    print(numero, end=" ")
```

Explicação: Leitura dos Números: O programa lê 10 números inteiros do usuário e os armazena na lista numeros. Duplicação dos Valores: O programa percorre a lista e duplica cada valor multiplicando-o por 2. Exibição do Array Duplicado: O programa exibe os números após a duplicação, separados por um espaço.

Exercício 100: Interseção de Dois Arrays

Enunciado: Desenvolva um programa que leia dois arrays de 10 números inteiros e exiba a interseção entre os dois arrays (os números que aparecem em ambos).

Código de solução:

```
# Preenchendo o array A
print("Preencha o array A:")
array_a = [int(input(f"Digite o número {i + 1}: ")) for i in range(10)]

# Preenchendo o array B
print("Preencha o array B:")
array_b = [int(input(f"Digite o número {i + 1}: ")) for i in range(10)]

# Encontrando a interseção
intersecao = [num for num in array_a if num in array_b]

# Exibindo a interseção
print("Interseção dos arrays:", intersecao)
```

Explicação: O programa lê dois arrays de 10 números inteiros cada e calcula a interseção, ou seja, os números que aparecem em ambos os arrays. Isso é feito percorrendo os elementos do primeiro array e verificando se cada elemento está presente no segundo. A complexidade de tempo é $O(n^2)$, pois, para cada elemento de um array, o programa verifica sua presença no segundo array. Caso os arrays sejam convertidos em conjuntos, a complexidade pode ser reduzida para $O(n)$, devido à eficiência das operações em conjuntos.

Exercício 101: União de Dois Arrays

Enunciado: Crie um programa que leia dois arrays de 10 números inteiros e exiba a união dos dois arrays (todos os números, sem repetição).

Código de solução:

```
# Lendo o primeiro array
print("Preencha o array A:")
array_a = [int(input(f"Digite o número {i + 1}: ")) for i in range(10)]

# Lendo o segundo array
print("Preencha o array B:")
```

```
array_b = [int(input(f"Digite o número {i + 1}: ")) for i in range(10)]

# Criando a união dos arrays sem repetição
uniao = list(set(array_a + array_b))

# Exibindo a união dos arrays
print("União dos arrays (sem repetição):", uniao)
```

Explicação: O programa lê dois arrays de 10 números inteiros.
Combina ambos os arrays em um único usando array_a + array_b.
Usa a função set para remover duplicatas, garantindo que a união não tenha repetições.
Converte o set de volta para list para exibir o resultado.
A complexidade de tempo é O(n), onde n é o número total de elementos, devido à eficiência das operações com conjuntos (set).

Exercício 102: Frequência de Elementos em um Array

Enunciado: Escreva um programa que leia 10 números inteiros e exiba a frequência de cada número, ou seja, quantas vezes cada número aparece no array.

Código de solução:

```
# Lendo os números e preenchendo o array
numeros = [int(input(f"Digite o número {i + 1}: ")) for i in range(10)]

# Calculando a frequência de cada número
frequencia = {}
for numero in numeros:
    frequencia[numero] = frequencia.get(numero, 0) + 1

# Exibindo a frequência de cada número
print("Frequência de cada número:")
for numero, contagem in frequencia.items():
    print(f"Número {numero} aparece {contagem} vez(es).")
```

Explicação: O programa lê 10 números inteiros e armazena em uma lista.
Utiliza um dicionário (frequencia) para contar quantas vezes cada número aparece.
O método get retorna o valor atual da chave ou 0 se a chave não existir.
A frequência de cada número é incrementada em 1 a cada ocorrência.
Exibe a frequência de cada número ao final.
A complexidade de tempo é O(n), onde n é a quantidade de números, pois cada número é inserido e contado uma vez.

Exercício 103: Separação de Pares e Ímpares

Enunciado: Desenvolva um programa que leia 10 números inteiros e separe os números pares dos números ímpares em dois arrays diferentes. Exiba os dois arrays.

Código de solução:

```
# Inicializando os arrays para pares e ímpares
pares = []
impares = []

# Lendo 10 números inteiros
for i in range(10):
    numero = int(input(f"Digite o número {i + 1}: "))
    if numero % 2 == 0:
        pares.append(numero)
    else:
        impares.append(numero)

# Exibindo os números pares
print("Números pares:")
print(" ".join(map(str, pares)))

# Exibindo os números ímpares
print("Números ímpares:")
print(" ".join(map(str, impares)))
```

Explicação: Inicialização: Dois arrays (listas) vazios são criados: um para armazenar os números pares e outro para ímpares.
Leitura e separação: O programa lê 10 números usando um laço for. Para cada número:
Se for divisível por 2 (numero % 2 == 0), é adicionado à lista de pares.
Caso contrário, é adicionado à lista de ímpares.
Exibição: Os números pares e ímpares são exibidos separadamente usando join para exibir os números em uma linha.

Exercício 104: Rotação de um Array

Enunciado: Crie um programa que leia 10 números inteiros e rotacione os elementos do array para a direita (o último elemento passa a ser o primeiro). Exiba o array após a rotação.

Código de solução:

```
# Inicializando o array de 10 números inteiros
numeros = []

# Preenchendo o array com 10 números fornecidos pelo usuário
for i in range(10):
    numero = int(input(f"Digite o número {i + 1}: "))
    numeros.append(numero)

# Rotacionando o array para a direita
ultimo_elemento = numeros[-1]  # Armazena o último elemento
numeros = [ultimo_elemento] + numeros[:-1]  # Coloca o último elemento
na frente

# Exibindo o array após a rotação
print("Array após a rotação:")
print(" ".join(map(str, numeros)))
```

Explicação: Preenchimento: O programa solicita ao usuário que insira 10 números, que são armazenados na lista numeros.
Rotação
O último elemento da lista é armazenado em ultimo_elemento.
A lista é recriada com o último elemento no início, seguido pelos demais elementos sem o último (numeros[:-1]).
Exibição: Os números são exibidos como uma string com espaços entre os elementos, usando join e map para converter os números em strings.

Conclusão do Capítulo 5: Arrays e Coleções

Neste capítulo, mergulhamos no estudo detalhado dos arrays unidimensionais e multidimensionais, explorando técnicas avançadas de manipulação. Durante os exercícios, os alunos foram desafiados a trabalhar com conceitos essenciais, como a declaração e inicialização de arrays, além de realizar operações básicas, incluindo iteração, busca e ordenação. A implementação de algoritmos clássicos, como Bubble Sort, Selection Sort e Insertion Sort, contribuiu para o desenvolvimento de habilidades importantes na resolução eficiente de problemas.

Nos arrays multidimensionais, práticas como o uso de matrizes para cálculos de soma, multiplicação e a identificação de elementos nas diagonais permitiram uma aplicação mais visual e profunda dos conceitos. Adicionalmente, a manipulação de arrays trouxe desafios avançados, como a busca binária, remoção e inserção de elementos, rotação de arrays, além de operações de interseção e união de arrays. Essa abordagem consolidou o entendimento e a aplicação prática de algoritmos e estruturas fundamentais na programação.

Capítulo 6: Métodos e Funções

Neste capítulo, exploraremos o conceito de funções, um elemento fundamental para a construção de programas modulares e reutilizáveis. Através dos exercícios propostos, você aprenderá a declarar e chamar funções, entender a passagem de parâmetros e como as funções retornam valores, além de descobrir técnicas que permitem criar múltiplas versões de uma mesma função para diferentes tipos ou quantidades de parâmetros.

Dividido em três seções, este capítulo começará com a construção e uso básico de funções, avançará para o conceito de parâmetros e valores de retorno, e finalizará com a adaptação das funções para aceitar diferentes formas de entrada. Isso permitirá que o aluno ganhe domínio sobre o poder das funções em programação orientada a objetos. Os exercícios serão desafiadores e práticos, reforçando a importância de modularizar o código e promover a reutilização de trechos, tornando os programas mais organizados e eficientes.

Declaração e Chamada de Métodos

Exercício 105: Saudação Simples

Enunciado: Crie um método chamado saudacao() que exiba a mensagem "Olá, seja bem-vindo!" no console. Em seguida, no método main(), chame este método para exibir a saudação.

Solução:

```
# Declaração do método saudacao
def saudacao():
    print("Olá, seja bem-vindo!")

# Função main para chamar o método saudacao
if __name__ == "__main__":
    saudacao()  # Chamada do método
```

Explicação: Neste exercício, o objetivo é criar e chamar uma função simples que exibe uma mensagem no console. A função saudacao() não recebe parâmetros e não retorna nenhum valor; sua única tarefa é exibir a mensagem "Olá, seja bem-vindo!". A função main() é responsável por chamar a função saudacao(). Esse tipo de função é útil para modularizar o código, tornando-o mais organizado e reutilizável.

Exercício 106: Exibir Número

Enunciado: Implemente um método chamado exibirNumero() que receba um número inteiro como parâmetro e exiba esse número no console. No método main(), chame o método passando diferentes valores para testar.

Solução:

```
# Declaração do método exibirNumero
def exibirNumero(numero):
    print(f"O número é: {numero}")

# Função main para testar o método com diferentes valores
if __name__ == "__main__":
    exibirNumero(5)   # Chamando o método com o valor 5
    exibirNumero(20)  # Chamando o método com o valor 20
```

Explicação: Neste exercício, o objetivo é criar uma função que recebe um número inteiro como parâmetro e o exibe no console. A função exibirNumero(numero) é chamada duas vezes no bloco principal (main()), passando valores diferentes (5 e 20), mostrando como um único método pode ser reutilizado para processar diferentes valores. O conceito de passar parâmetros para funções permite que o código seja mais flexível e modular.

Exercício 107: Soma de Dois Números

Enunciado: Crie um método chamado somar() que receba dois números inteiros como parâmetros, calcule a soma deles e exiba o resultado no console. No método main(), solicite dois números do usuário, chame o método e exiba o resultado.

Solução:

```
# Declaração do método somar
def somar(a, b):
    soma = a + b
    print(f"A soma é: {soma}")

# Função main para solicitar os números e chamar o método
if __name__ == "__main__":
    num1 = int(input("Digite o primeiro número: "))
    num2 = int(input("Digite o segundo número: "))

    somar(num1, num2)  # Chamando o método somar
```

Explicação: Neste exercício, o objetivo é criar uma função somar() que recebe dois números inteiros como parâmetros, calcula a soma deles e exibe o resultado. A interação com o usuário ocorre no bloco principal, onde os números são solicitados com input() e, em seguida, passados como argumentos para o método somar(). Esse exercício mostra como separar a lógica de cálculo da interação com o usuário, melhorando a organização e legibilidade do código.

Exercício 108: Verificação de Paridade

Enunciado: Implemente um método chamado verificarParidade() que receba um número inteiro como parâmetro e exiba se o número é par ou ímpar. No método main(), chame o método para testar com diferentes números.

Solução:

```
# Declaração do método verificarParidade
def verificar_paridade(numero):
    if numero % 2 == 0:
        print(f"O número {numero} é par.")
    else:
        print(f"O número {numero} é ímpar.")

# Função main para testar com diferentes números
if __name__ == "__main__":
    verificar_paridade(7)  # Chamando o método com o valor 7
    verificar_paridade(10) # Chamando o método com o valor 10
```

Explicação: Neste exercício, criamos a função verificar_paridade(), que recebe um número inteiro como parâmetro e verifica se ele é par ou ímpar. O operador % (módulo) é utilizado para calcular o resto da divisão por 2, e dependendo do resultado, a função exibe se o número é par ou ímpar. O uso dessa função facilita a reutilização do código para diferentes números em diferentes partes do programa.

Exercício 109: Cálculo de Área de Retângulo

Enunciado: Desenvolva um método chamado calcularAreaRetangulo() que receba a largura e a altura de um retângulo como parâmetros e exiba a área no console. No método main(), solicite os valores ao usuário, chame o método e exiba o resultado.

Solução:

```
# Declaração do método calcular_area_retangulo
```

```
def calcular_area_retangulo(largura, altura):
    area = largura * altura
    print(f"A área do retângulo é: {area}")

# Função main para solicitar os valores ao usuário e chamar o método
if __name__ == "__main__":
    largura = float(input("Digite a largura do retângulo: "))
    altura = float(input("Digite a altura do retângulo: "))

    calcular_area_retangulo(largura, altura)  # Chamada do método
```

Explicação: Neste exercício, a função calcular_area_retangulo() recebe dois parâmetros do tipo float, que representam a largura e a altura do retângulo. Ela calcula a área multiplicando esses valores e exibe o resultado. A interação com o usuário ocorre na função principal, onde solicitamos os valores de entrada. Essa separação de responsabilidades torna o código mais modular, legível e reutilizável. O uso de float permite trabalhar com valores decimais, oferecendo maior flexibilidade no cálculo.

Passagem de Parâmetros e Retorno de Valores

Exercício 110: Retorno de Saudação

Enunciado: Crie um método chamado obterSaudacao() que retorne a string "Olá, seja bem-vindo!". No método main(), armazene o valor retornado em uma variável e exiba a saudação.

Solução:

```
# Declaração do método obter_saudacao
def obter_saudacao():
    return "Olá, seja bem-vindo!"

# Função principal
if __name__ == "__main__":
    saudacao = obter_saudacao()  # Chamando o método e armazenando o
retorno
    print(saudacao)              # Exibindo a saudação
```

Explicação: Neste exercício, a função obter_saudacao() retorna uma string contendo a mensagem de saudação. No bloco principal, chamamos essa função e armazenamos o valor retornado na variável saudacao. Em seguida, exibimos a mensagem usando a função

print(). Esse exemplo demonstra como o retorno de valores permite que os métodos (ou funções) sejam reutilizados em diferentes contextos, promovendo uma maior flexibilidade no código.

Exercício 111: Soma com Retorno

Enunciado: Implemente um método chamado somar() que receba dois números inteiros como parâmetros, calcule a soma e retorne o resultado. No método main(), capture o valor retornado e exiba-o.

Solução:

```
# Declaração do método somar
def somar(a, b):
    return a + b  # Retornando a soma

# Função principal
if __name__ == "__main__":
    resultado = somar(10, 5)  # Chamada do método somar com retorno
    print("A soma é:", resultado)  # Exibindo o resultado
```

Explicação: A função somar(a, b) recebe dois parâmetros inteiros, calcula a soma e retorna o resultado. No bloco principal (if __name__ == "__main__":), chamamos a função somar passando os valores 10 e 5 como argumentos. O valor retornado é armazenado na variável resultado e exibido no console usando a função print().

Esse exercício destaca a importância do retorno de valores, permitindo que o resultado da função seja utilizado em diferentes partes do programa. Isso contribui para a reutilização de código e melhora a modularização.

Exercício 112: Verificar Paridade com Retorno

Enunciado: Desenvolva um método chamado ehPar() que receba um número inteiro como parâmetro e retorne true se o número for par e false se for ímpar. No método main(), utilize o valor retornado para exibir a mensagem correspondente.

Solução:

```
# Declaração do método eh_par
def eh_par(numero):
    return numero % 2 == 0  # Retornando True se for par
```

```
# Função principal
if __name__ == "__main__":
    numero = 10  # Exemplo de número a ser verificado
    resultado = eh_par(numero)  # Chamada do método eh_par com retorno
    if resultado:
        print("O número é par.")
    else:
        print("O número é ímpar.")
```

Explicação:A função eh_par(numero) recebe um número inteiro como parâmetro e retorna True se o número for par (ou seja, se o resto da divisão por 2 for 0) ou False se for ímpar. No bloco principal, chamamos a função com o valor 10, capturamos o retorno em resultado e usamos uma condicional para exibir a mensagem correta.

Exercício 113: Cálculo de Área com Retorno

Enunciado: Escreva um método chamado calcularAreaRetangulo() que receba a largura e a altura de um retângulo e retorne a área calculada. No método main(), exiba a área retornada.

Solução:

```
# Declaração do método calcular_area_retangulo
def calcular_area_retangulo(largura, altura):
    return largura * altura  # Retornando a área do retângulo

# Função principal
if __name__ == "__main__":
    largura = 5.0
    altura = 3.0
    area = calcular_area_retangulo(largura, altura)  # Chamada do método
com retorno
    print(f"A área do retângulo é: {area}")
```

Explicação: A função calcular_area_retangulo(largura, altura) recebe dois parâmetros (largura e altura), calcula a área multiplicando os valores e retorna o resultado. No bloco principal, chamamos a função com os valores 5.0 e 3.0, capturamos o retorno em area e exibimos o resultado no console usando print.

Exercício 114: Potenciação com Retorno

Enunciado: Implemente um método chamado potencia() que receba dois números inteiros (base e expoente) e retorne o resultado da potenciação. No método main(), capture e exiba o valor retornado.

Solução:

```
# Declaração do método potencia
def potencia(base, expoente):
    resultado = 1
    for _ in range(expoente):  # Loop para multiplicar a base pelo
número de vezes do expoente
        resultado *= base
    return resultado

# Função principal
if __name__ == "__main__":
    base = 2
    expoente = 3
    resultado = potencia(base, expoente)  # Chamada do método potencia
    print(f"O resultado da potenciação é: {resultado}")
```

Explicação: A função potencia(base, expoente) recebe dois números inteiros como parâmetros e calcula a exponenciação usando um loop for, multiplicando a base pelo número de vezes correspondente ao valor do expoente. O valor final é retornado e exibido no bloco principal.

Exercício 115: Conversão de Temperatura

Enunciado: Desenvolva um método chamado converterCelsiusParaFahrenheit() que receba uma temperatura em graus Celsius e retorne o valor em Fahrenheit. No método main(), exiba a temperatura convertida.

Solução:

```
# Declaração do método converterCelsiusParaFahrenheit
def converter_celsius_para_fahrenheit(celsius):
    return (celsius * 9 / 5) + 32  # Fórmula de conversão de Celsius
para Fahrenheit

# Função principal
if __name__ == "__main__":
    celsius = 25.0
    fahrenheit = converter_celsius_para_fahrenheit(celsius)  # Chamada
do método com retorno
```

```
    print(f"A temperatura em Fahrenheit é: {fahrenheit}")
```

Explicação: A função converter_celsius_para_fahrenheit(celsius) recebe a temperatura em graus Celsius, aplica a fórmula de conversão para Fahrenheit e retorna o resultado. No bloco principal, chamamos essa função com a temperatura de 25°C, armazenamos o resultado na variável fahrenheit e o exibimos.

Exercício 116: Verificar Maior Número

Enunciado: Crie um método chamado obterMaior() que receba dois números inteiros e retorne o maior deles. No método main(), capture o valor retornado e exiba o maior número.

Solução:

```
# Declaração do método obter_maior
def obter_maior(num1, num2):
    if num1 > num2:
        return num1
    else:
        return num2

# Função principal
if __name__ == "__main__":
    maior = obter_maior(10, 20)  # Chamada do método obter_maior
    print(f"O maior número é: {maior}")
```

Explicação: A função obter_maior(num1, num2) recebe dois números inteiros, compara-os usando uma estrutura condicional if-else e retorna o maior valor. No bloco principal, chamamos a função com os valores 10 e 20, armazenamos o resultado na variável maior e o exibimos. Esse tipo de implementação é útil para comparar valores rapidamente e retornar o maior deles.

Exercício 117: Média Aritmética com Retorno

Enunciado: Escreva um método chamado calcularMedia() que receba três números inteiros e retorne a média aritmética deles. No método main(), exiba a média retornada.

Solução:

```
# Declaração do método calcular_media
```

```
def calcular_media(num1, num2, num3):
    return (num1 + num2 + num3) / 3.0  # Calculando e retornando a média

# Função principal
if __name__ == "__main__":
    media = calcular_media(10, 20, 30)  # Chamada do método
calcular_media
    print(f"A média é: {media}")
```

Explicação: A função calcular_media(num1, num2, num3) recebe três números inteiros, soma os valores e divide por 3.0 para garantir que o resultado seja um valor com casas decimais. No bloco principal, a função é chamada com os números 10, 20 e 30, e o resultado é armazenado na variável media, que é exibida em seguida. Esse método encapsula a lógica do cálculo da média, tornando o código mais modular e reutilizável.

Exercício 118: Verificar Letra Maiúscula

Enunciado: Desenvolva um método chamado ehMaiuscula() que receba um caractere e retorne true se for uma letra maiúscula, e false se for uma letra minúscula. No método main(), utilize o valor retornado para exibir a mensagem correspondente.

Solução:

```
# Declaração do método eh_maiuscula
def eh_maiuscula(caractere):
    return caractere.isupper()  # Verifica se o caractere é maiúsculo

# Função principal
if __name__ == "__main__":
    resultado = eh_maiuscula('A')  # Chamada do método eh_maiuscula
    if resultado:
        print("O caractere é maiúsculo.")
    else:
        print("O caractere é minúsculo.")
```

Explicação: A função eh_maiuscula(caractere) usa o método isupper() da classe str para verificar se o caractere recebido é maiúsculo, retornando True ou False conforme o caso. No bloco principal, a função é chamada com o caractere 'A', e o valor retornado é usado para exibir a mensagem correspondente. Essa abordagem separa a lógica de verificação do código principal, tornando-o mais organizado e reutilizável.

Exercício 119: Raiz Quadrada com Retorno

Enunciado: Implemente um método chamado calcularRaizQuadrada() que receba um número inteiro e retorne a raiz quadrada desse número. No método main(), capture e exiba o valor retornado.

Solução:

```
import math  # Importa a biblioteca para cálculos matemáticos

# Declaração do método calcular_raiz_quadrada
def calcular_raiz_quadrada(numero):
    return math.sqrt(numero)  # Calcula e retorna a raiz quadrada

# Função principal
if __name__ == "__main__":
    raiz = calcular_raiz_quadrada(16)  # Chamada do método
calcular_raiz_quadrada
    print(f"A raiz quadrada é: {raiz}")
```

Explicação: A função calcular_raiz_quadrada(numero) utiliza o método math.sqrt() da biblioteca padrão math para calcular a raiz quadrada do número recebido. O resultado é então retornado e exibido no bloco principal. Esse exercício ilustra como utilizar funções nativas da linguagem para cálculos complexos, promovendo a reutilização de código.

Exercício 120: Calcular Fatorial

Enunciado: Crie um método chamado calcularFatorial() que receba um número inteiro e retorne o fatorial desse número. No método main(), exiba o valor retornado.

Solução:

```
# Declaração do método calcular_fatorial
def calcular_fatorial(numero):
    resultado = 1
    for i in range(1, numero + 1):
        resultado *= i  # Multiplicando os valores sucessivos
    return resultado  # Retornando o fatorial

# Função principal
if __name__ == "__main__":
    fatorial = calcular_fatorial(5)  # Chamada do método
calcular_fatorial
    print(f"O fatorial é: {fatorial}")
```

Explicação: A função calcular_fatorial(numero) usa um loop for que vai de 1 até o valor do número (inclusivo). A cada iteração, o valor de resultado é multiplicado pelo contador i, acumulando o produto até atingir o número desejado. O valor final é retornado e exibido no bloco principal. Esse tipo de implementação demonstra como usar loops para resolver problemas iterativos, encapsulando a lógica de cálculo em uma função reutilizável.

Exercício 121: Verificar Número Primo

Enunciado: Desenvolva um método chamado ehPrimo() que receba um número inteiro e retorne true se o número for primo, e false caso contrário. No método main(), utilize o valor retornado para exibir a mensagem correspondente.

Solução:

```
# Declaração do método eh_primo
import math

def eh_primo(numero):
    if numero < 2:
        return False  # Números menores que 2 não são primos
    for i in range(2, int(math.sqrt(numero)) + 1):
        if numero % i == 0:
            return False  # Se divisível por qualquer número além de 1 e
ele mesmo
    return True  # Se não encontrou nenhum divisor, o número é primo

# Função principal
if __name__ == "__main__":
    resultado = eh_primo(7)  # Chamada do método eh_primo
    if resultado:
        print("O número é primo.")
    else:
        print("O número não é primo.")
```

Explicação: A função eh_primo(numero) verifica se o número é primo. Se o número for menor que 2, retorna False. Caso contrário, ela verifica se o número é divisível por algum número entre 2 e a raiz quadrada de numero. Se for divisível, retorna False, indicando que não é primo. Se não encontrar nenhum divisor, retorna True, indicando que o número é primo. A função principal exibe a mensagem apropriada com base no resultado retornado.

Exercício 122: Contagem de Vogais

Enunciado: Implemente um método chamado contarVogais() que receba uma string como parâmetro e retorne a quantidade de vogais presentes na string. No método main(), exiba o valor retornado.

Solução:

```
# Declaração do método contar_vogais
def contar_vogais(texto):
    contagem = 0
    vogais = "aeiouAEIOU"  # Definindo as vogais
    for char in texto:
        if char in vogais:  # Verifica se o caractere é uma vogal
            contagem += 1
    return contagem  # Retorna o número total de vogais

# Função principal
if __name__ == "__main__":
    total_vogais = contar_vogais("Programacao")  # Chamada do método contar_vogais
    print(f"O total de vogais é: {total_vogais}")
```

Explicação: A função contar_vogais(texto) percorre cada caractere da string texto e verifica se ele está presente na string vogais (que contém todas as vogais, tanto maiúsculas quanto minúsculas). Para cada vogal encontrada, a variável contagem é incrementada. Ao final, a função retorna a quantidade de vogais encontradas. No método principal, o valor retornado é exibido.

Exercício 123: Conversão de Moedas

Enunciado: Crie um método chamado converterRealParaDolar() que receba um valor em reais e retorne o valor convertido em dólares. No método main(), exiba o valor convertido. Considere a taxa de câmbio fixa de 1 dólar = 5,00 reais.

Solução:

```
# Declaração do método converter_real_para_dolar
def converter_real_para_dolar(valor_real):
    taxa_cambio = 5.0  # Taxa de câmbio fixa
    return valor_real / taxa_cambio  # Retorna o valor convertido para dólares

# Função principal
```

```
if __name__ == "__main__":
    valor_em_dolar = converter_real_para_dolar(50.0)  # Chamada do
método converter_real_para_dolar
    print(f"O valor em dólares é: {valor_em_dolar}")
```

Explicação: A função converter_real_para_dolar(valor_real) recebe um valor em reais e o divide pela taxa de câmbio (5.0 reais para 1 dólar), retornando o valor convertido para dólares. No método principal, o valor retornado pela função é exibido na tela.

Exercício 124: Menor de Três Números

Enunciado: Desenvolva um método chamado obterMenor() que receba três números inteiros e retorne o menor deles. No método main(), capture o valor retornado e exiba o menor número.

Solução:

```
# Função para obter o menor número entre três
def obter_menor(num1, num2, num3):
    menor = num1
    if num2 < menor:
        menor = num2
    if num3 < menor:
        menor = num3
    return menor  # Retorna o menor número

# Função principal
if __name__ == "__main__":
    menor = obter_menor(15, 7, 10)  # Chamada do método obter_menor
    print(f"O menor número é: {menor}")
```

Explicação: A função obter_menor(num1, num2, num3) compara três números e retorna o menor entre eles. Inicializa a variável menor com o primeiro número e a compara com os outros dois números, atualizando o valor de menor quando encontrar um número menor. O código no bloco if __name__ == "__main__": chama a função e exibe o menor número encontrado.

Sobrecarga de métodos

Exercício 125: Sobrecarga do Método Soma

Enunciado: Implemente uma classe com dois métodos chamados somar(). O primeiro método deve receber dois números inteiros como parâmetros e retornar a soma deles. O segundo método deve receber três números inteiros e retornar a soma dos três. No método main(), teste os dois métodos chamando-os com diferentes quantidades de parâmetros.

Solução:

```
class SobrecargaSoma:

    # Método para somar dois números
    def somar(self, num1, num2):
        return num1 + num2

    # Método para somar três números
    def somar_tres(self, num1, num2, num3):
        return num1 + num2 + num3

# Função principal
if __name__ == "__main__":
    soma = SobrecargaSoma()
    print("Soma de dois números:", soma.somar(5, 10))  # Chamando o
método com 2 parâmetros
    print("Soma de três números:", soma.somar_tres(3, 7, 2))  # Chamando
o método com 3 parâmetros
```

Explicação: A classe SobrecargaSoma contém dois métodos de soma. Como o Python não suporta sobrecarga de métodos diretamente (métodos com o mesmo nome mas diferentes parâmetros), usamos métodos distintos: somar() para somar dois números e somar_tres() para somar três números.
No bloco if __name__ == "__main__":, criamos uma instância da classe SobrecargaSoma e chamamos ambos os métodos com diferentes quantidades de parâmetros.

Exercício 126: Sobrecarga do Método Calcular Área

Enunciado: Desenvolva uma classe que contenha dois métodos sobrecarregados chamados calcularArea(). O primeiro método deve calcular e retornar a área de um quadrado, recebendo o comprimento de um dos lados como parâmetro. O segundo método deve calcular e retornar a área de um retângulo, recebendo a largura e a altura como parâmetros. No método main(), teste os dois métodos.

Solução:

```
class SobrecargaArea:

    # Método para calcular a área de um quadrado
    def calcular_area_quadrado(self, lado):
        return lado * lado

    # Método para calcular a área de um retângulo
    def calcular_area_retangulo(self, largura, altura):
        return largura * altura

# Função principal
if __name__ == "__main__":
    area = SobrecargaArea()
    print("Área do quadrado:", area.calcular_area_quadrado(5))  #
Chamando o método com 1 parâmetro
    print("Área do retângulo:", area.calcular_area_retangulo(5, 10))  #
Chamando o método com 2 parâmetros
```

Explicação: Em Python, como não há suporte direto para sobrecarga de métodos (métodos com o mesmo nome mas diferentes parâmetros), criamos dois métodos com nomes diferentes: calcular_area_quadrado() e calcular_area_retangulo().
O método calcular_area_quadrado() calcula a área do quadrado com base no lado, e o método calcular_area_retangulo() calcula a área do retângulo com base na largura e altura.
No bloco if __name__ == "__main__":, instanciamos a classe SobrecargaArea e chamamos os métodos para calcular as áreas de um quadrado e um retângulo.

Exercício 127: Sobrecarga do Método Exibir Mensagem

Enunciado: Crie uma classe com dois métodos sobrecarregados chamados exibirMensagem(). O primeiro método deve exibir uma mensagem padrão no console. O segundo método deve receber uma string como parâmetro e exibir essa string no console. No método main(), chame ambos os métodos para exibir as mensagens.

Solução:

```
class SobrecargaMensagem:

    # Método para exibir a mensagem padrão
    def exibir_mensagem(self):
        print("Mensagem padrão: Olá, mundo!")

    # Método para exibir uma mensagem personalizada
```

```
    def exibir_mensagem_personalizada(self, mensagem):
        print(f"Mensagem personalizada: {mensagem}")

# Função principal
if __name__ == "__main__":
    mensagem = SobrecargaMensagem()
    mensagem.exibir_mensagem()  # Chamando o método sem parâmetros
    mensagem.exibir_mensagem_personalizada("Bem-vindo!")  # Chamando o
método com uma string como parâmetro
```

Explicação: Em Python, não existe sobrecarga de métodos diretamente, então criamos dois métodos diferentes com nomes distintos: exibir_mensagem() para a mensagem padrão e exibir_mensagem_personalizada() para a mensagem personalizada.
O método exibir_mensagem() exibe a mensagem padrão "Olá, mundo!", enquanto o método exibir_mensagem_personalizada() exibe a mensagem personalizada passada como argumento.
No bloco if __name__ == "__main__":, instanciamos a classe SobrecargaMensagem e chamamos ambos os métodos para exibir as mensagens.

Conclusão do Capítulo 6: Métodos e Funções

Neste capítulo, exploramos os principais conceitos de métodos e funções em Python, passando pela criação, uso e simulação de sobrecarga de métodos.

Os exercícios proporcionaram prática na definição e chamada de funções, mostrando como modularizar o código, tornando-o mais organizado e reutilizável. Na seção de passagem de parâmetros e retorno de valores, aprendemos a enviar informações para as funções e receber respostas, o que possibilita realizar cálculos e manipulações de forma eficiente.

Por fim, na simulação de sobrecarga de métodos, vimos como utilizar o mesmo nome de função com diferentes comportamentos para lidar com diferentes tipos de operações.

Introdução ao Capítulo 7: Classes e Objetos

Neste capítulo, vamos explorar um dos pilares fundamentais da programação orientada a objetos (POO): a criação e manipulação de classes e objetos. Através de exemplos práticos, você aprenderá a definir classes, que são os modelos para criar objetos, e a compreender como os atributos e métodos de instância moldam o comportamento desses objetos no programa.

Você também verá como o encapsulamento, um dos conceitos mais importantes da POO, permite proteger os dados de uma classe, controlando seu acesso e modificação por meio

de getters e setters. O encapsulamento promove segurança e flexibilidade, assegurando que os atributos sejam manipulados de maneira adequada.

Definição de Classes e Objetos

Exercício 128: Definindo uma Classe Simples

Enunciado: Crie uma classe chamada Carro. Defina três atributos para a classe: marca, modelo e ano. Em seguida, crie um objeto dessa classe no método main() e inicialize os atributos com valores.

Solução:

```
class Carro:
    def __init__(self, marca, modelo, ano):
        self.marca = marca
        self.modelo = modelo
        self.ano = ano

# Criando o objeto Carro no método principal
if __name__ == "__main__":
    meu_carro = Carro("Toyota", "Corolla", 2020)  # Inicializando os
atributos

    # Exibindo os valores dos atributos
    print(f"Marca: {meu_carro.marca}")
    print(f"Modelo: {meu_carro.modelo}")
    print(f"Ano: {meu_carro.ano}")
```

Explicação: Criamos a classe Carro com três atributos: marca, modelo e ano.
O método __init__ é usado para inicializar esses atributos ao criar um objeto.
No bloco if __name__ == "__main__", instanciamos a classe com valores específicos e exibimos os atributos do objeto meu_carro.

Exercício 129: Definindo Atributos e Métodos

Enunciado: Crie uma classe chamada Pessoa com os atributos nome e idade. Adicione um método chamado apresentar() que exiba no console: "Olá, meu nome é [nome] e tenho [idade] anos". No método main(), crie dois objetos Pessoa e chame o método apresentar() para ambos.

Solução:

```
class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade

    # Método para apresentação
    def apresentar(self):
        print(f"Olá, meu nome é {self.nome} e tenho {self.idade} anos.")

# Método principal
if __name__ == "__main__":
    pessoa1 = Pessoa("Carlos", 30)  # Criando a primeira instância de
Pessoa
    pessoa2 = Pessoa("Ana", 25)    # Criando a segunda instância de
Pessoa

    pessoa1.apresentar()  # Chamada do método apresentar para pessoa1
    pessoa2.apresentar()  # Chamada do método apresentar para pessoa2
```

Explicação: Criamos a classe Pessoa com dois atributos: nome e idade, inicializados no método __init__. O método apresentar() exibe uma mensagem formatada utilizando os atributos da instância. No bloco if __name__ == "__main__", criamos duas instâncias da classe Pessoa com diferentes valores para nome e idade. Chamamos o método apresentar() em cada instância para exibir a mensagem personalizada.

Exercício 130: Definindo uma Classe com Construtor

Enunciado: Implemente uma classe chamada ContaBancaria com os atributos numeroConta e saldo. Crie um construtor que inicialize esses atributos ao criar um objeto. No método main(), crie uma conta bancária usando o construtor.

Solução:

```
class ContaBancaria:
    def __init__(self, numero_conta, saldo_inicial):
        self.numero_conta = numero_conta
        self.saldo = saldo_inicial

# Método principal
if __name__ == "__main__":
```

```
    minha_conta = ContaBancaria(12345, 500.0)  # Criando uma conta
bancária
    print(f"Número da conta: {minha_conta.numero_conta}")
    print(f"Saldo inicial: {minha_conta.saldo}")
```

Explicação: A classe ContaBancaria possui o método __init__, que é o construtor em Python. Ele é usado para inicializar os atributos numero_conta e saldo no momento em que o objeto é criado. No bloco if __name__ == "__main__", criamos uma instância da classe ContaBancaria, passando o número da conta e o saldo inicial como argumentos para o construtor. Por fim, os valores dos atributos são exibidos usando print.

Exercício 131: Métodos para Manipular Dados

Enunciado: Adicione à classe ContaBancaria os métodos depositar(double valor) e sacar(double valor), que atualizam o saldo da conta. No método main(), simule um depósito e um saque, e exiba o saldo após cada operação.

Solução:

```
class ContaBancaria:
    def __init__(self, numero_conta, saldo_inicial):
        self.numero_conta = numero_conta
        self.saldo = saldo_inicial

    # Método para depósito
    def depositar(self, valor):
        self.saldo += valor

    # Método para saque
    def sacar(self, valor):
        if valor <= self.saldo:
            self.saldo -= valor
        else:
            print("Saldo insuficiente.")

# Método principal
if __name__ == "__main__":
    minha_conta = ContaBancaria(12345, 500.0)

    # Simulando operações de depósito e saque
    minha_conta.depositar(200.0)
    print(f"Saldo após depósito: {minha_conta.saldo}")

    minha_conta.sacar(100.0)
```

```
    print(f"Saldo após saque: {minha_conta.saldo}")
```

Explicação:
Construtor __init__:
Inicializa os atributos numero_conta e saldo com os valores fornecidos ao criar um objeto da classe.
Método depositar:
Adiciona o valor fornecido ao saldo atual da conta.
Método sacar:
Verifica se o valor a ser sacado é menor ou igual ao saldo. Se for, deduz o valor do saldo. Caso contrário, exibe uma mensagem informando que o saldo é insuficiente.
Execução no bloco principal:
Criamos um objeto minha_conta e inicializamos com um número de conta e saldo inicial.
Realizamos operações de depósito e saque, exibindo o saldo após cada operação.

Exercício 132: Classe Produto

Enunciado: Crie uma classe chamada Produto com os atributos nome, preco e quantidade. Adicione um método para calcular o valor total do estoque (preco * quantidade) e outro método para exibir os detalhes do produto. No método main(), crie dois objetos e teste os métodos.

Solução:

```
class Produto:
    def __init__(self, nome, preco, quantidade):
        self.nome = nome
        self.preco = preco
        self.quantidade = quantidade

    # Método para calcular o valor total do estoque
    def calcular_valor_estoque(self):
        return self.preco * self.quantidade

    # Método para exibir detalhes do produto
    def exibir_detalhes(self):
        print(f"Produto: {self.nome}")
        print(f"Preço: R$ {self.preco:.2f}")
        print(f"Quantidade em estoque: {self.quantidade}")
        print(f"Valor total em estoque: R$
{self.calcular_valor_estoque():.2f}")

# Método principal
```

```
if __name__ == "__main__":
    produto1 = Produto("Notebook", 3000.0, 10)
    produto2 = Produto("Mouse", 50.0, 100)

    # Exibindo detalhes dos produtos
    produto1.exibir_detalhes()
    produto2.exibir_detalhes()
```

Explicação:
Construtor __init__:
Inicializa os atributos nome, preco e quantidade ao criar um objeto da classe.
Método calcular_valor_estoque:
Calcula o valor total do estoque multiplicando o preço pela quantidade.
Método exibir_detalhes:
Exibe os detalhes do produto, como nome, preço, quantidade em estoque e o valor total calculado do estoque.
Os valores numéricos são formatados para mostrar duas casas decimais.
Execução no bloco principal:
Criamos dois objetos (produto1 e produto2) com valores diferentes para seus atributos.
Chamamos o método exibir_detalhes para cada produto, mostrando suas informações e o valor total do estoque.

Exercício 133: Relacionamento entre Classes

Enunciado: Crie duas classes: Aluno e Turma. A classe Turma deve ter o atributo nome e um método para exibir o nome da turma. A classe Aluno deve ter os atributos nome e turma, sendo turma do tipo Turma. No método main(), crie objetos das duas classes e exiba as informações.

Solução:

```
class Turma:
    def __init__(self, nome):
        self.nome = nome

    # Método para exibir o nome da turma
    def exibir_turma(self):
        print(f"Turma: {self.nome}")


class Aluno:
    def __init__(self, nome, turma):
```

```
        self.nome = nome
        self.turma = turma  # Atributo turma do tipo Turma

    # Método para exibir os dados do aluno
    def exibir_dados(self):
        print(f"Aluno: {self.nome}")
        self.turma.exibir_turma()  # Chama o método da classe Turma


# Método principal
if __name__ == "__main__":
    turma1 = Turma("3A")  # Criando um objeto Turma
    aluno1 = Aluno("João", turma1)  # Relacionando o aluno à turma

    # Exibindo os dados do aluno e da turma
    aluno1.exibir_dados()
```

Explicação:
Classe Turma:
Contém o atributo nome, que armazena o nome da turma.
Possui o método exibir_turma, que exibe o nome da turma.
Classe Aluno:
Contém os atributos nome (nome do aluno) e turma (um objeto da classe Turma).
O método exibir_dados exibe o nome do aluno e chama o método exibir_turma da turma associada.
Execução no bloco principal:
Criamos um objeto da classe Turma chamado turma1 com o nome "3A".
Criamos um objeto da classe Aluno chamado aluno1, relacionando-o à turma criada.
O método exibir_dados exibe as informações do aluno e da turma, mostrando o relacionamento entre os objetos.

Exercício 134: Classe Retângulo

Enunciado: Implemente uma classe Retangulo com os atributos largura e altura. Adicione métodos para calcular a área e o perímetro do retângulo. No método main(), crie um objeto Retangulo e exiba seus valores de área e perímetro.

Solução:

```
class Retangulo:
    def __init__(self, largura, altura):
        self.largura = largura
```

```
        self.altura = altura

    # Método para calcular a área do retângulo
    def calcular_area(self):
        return self.largura * self.altura

    # Método para calcular o perímetro do retângulo
    def calcular_perimetro(self):
        return 2 * (self.largura + self.altura)


# Método principal
if __name__ == "__main__":
    # Criando um objeto Retangulo
    retangulo = Retangulo(5.0, 3.0)

    # Exibindo os valores de área e perímetro
    print(f"Área: {retangulo.calcular_area()}")
    print(f"Perímetro: {retangulo.calcular_perimetro()}")
```

Explicação:A classe Retangulo possui dois atributos: largura e altura, que representam as dimensões do retângulo. O construtor __init__ é usado para inicializar esses atributos ao criar um objeto.
Dentro da classe, temos dois métodos principais:
calcular_area: Esse método retorna o valor da área do retângulo, que é obtido multiplicando a largura pela altura. calcular_perimetro: Esse método calcula o perímetro do retângulo, que é dado pela fórmula 2 * (largura + altura). No bloco principal, criamos um objeto retangulo, onde atribuímos valores para a largura e altura. Em seguida, chamamos os métodos para calcular a área e o perímetro, e exibimos esses valores no console.

Exercício 135: Definindo Métodos com Retorno

Enunciado: Crie uma classe Cilindro com os atributos raio e altura. Adicione um método calcularVolume() que retorne o volume do cilindro (π * raio² * altura). No método main(), crie um objeto Cilindro e exiba o volume.

Solução:

```
import math

class Cilindro:
    def __init__(self, raio, altura):
        self.raio = raio
```

```
        self.altura = altura

    # Método para calcular o volume do cilindro
    def calcular_volume(self):
        return math.pi * (self.raio ** 2) * self.altura

def main():
    cilindro = Cilindro(3.0, 10.0)

    print(f"Volume do cilindro: {cilindro.calcular_volume()}")

if __name__ == "__main__":
    main()
```

Explicação: A classe Cilindro tem dois atributos: raio e altura, que representam as dimensões do cilindro. O método calcular_volume() utiliza a fórmula matemática para calcular o volume do cilindro (π * raio² * altura) e retorna o valor calculado. No método main(), criamos um objeto da classe Cilindro, passando valores para raio e altura. Em seguida, chamamos o método calcular_volume() para exibir o volume do cilindro. A biblioteca math é usada para acessar o valor de π (pi) e fazer os cálculos necessários.

Exercício 136: Classe Aluno com Média

Enunciado: Implemente uma classe Aluno com os atributos nome, nota1 e nota2. Adicione um método calcularMedia() que calcule e retorne a média das duas notas. No método main(), crie um aluno e exiba sua média.

Solução:

```
class Aluno:
    def __init__(self, nome, nota1, nota2):
        self.nome = nome
        self.nota1 = nota1
        self.nota2 = nota2

    # Método para calcular a média das notas
    def calcular_media(self):
        return (self.nota1 + self.nota2) / 2

def main():
    aluno = Aluno("Carlos", 8.0, 9.5)
    print(f"Média do aluno {aluno.nome}: {aluno.calcular_media()}")
```

```
if __name__ == "__main__":
    main()
```

Explicação: A classe Aluno contém três atributos: nome, nota1 e nota2, que armazenam o nome e as notas do aluno. O método calcular_media() calcula e retorna a média das duas notas do aluno. No método main(), criamos um objeto Aluno com o nome "Carlos" e notas 8.0 e 9.5. Depois, chamamos o método calcular_media() para exibir a média do aluno.
O uso de f-strings no Python permite uma forma mais concisa e legível para imprimir a média do aluno.

Exercício 137: Classe com Métodos Estáticos

Enunciado: Crie uma classe Calculadora com métodos estáticos somar(), subtrair(), multiplicar(), dividir(), que realizam as operações matemáticas básicas. No método main(), chame esses métodos sem criar um objeto.

Solução:

```
class Calculadora:

    # Método estático para soma
    @staticmethod
    def somar(a, b):
        return a + b

    # Método estático para subtração
    @staticmethod
    def subtrair(a, b):
        return a - b

    # Método estático para multiplicação
    @staticmethod
    def multiplicar(a, b):
        return a * b

    # Método estático para divisão
    @staticmethod
    def dividir(a, b):
        if b != 0:
            return a / b
        else:
            print("Erro: Divisão por zero.")
            return 0
```

```
def main():
    print(f"Soma: {Calculadora.somar(10, 5)}")
    print(f"Subtração: {Calculadora.subtrair(10, 5)}")
    print(f"Multiplicação: {Calculadora.multiplicar(10, 5)}")
    print(f"Divisão: {Calculadora.dividir(10, 5)}")

if __name__ == "__main__":
    main()
```

Explicação: A classe Calculadora possui métodos estáticos, que são definidos com o decorador @staticmethod no Python. Esses métodos podem ser chamados diretamente pela classe sem a necessidade de criar uma instância. Os métodos somar(), subtrair(), multiplicar() e dividir() realizam as operações matemáticas básicas. No main(), chamamos diretamente os métodos estáticos da classe Calculadora para exibir os resultados das operações. O método dividir() também trata a divisão por zero, exibindo uma mensagem de erro caso o divisor seja zero.

Exercício 138: Classe Livro

Enunciado: Implemente uma classe Livro com os atributos titulo, autor e numeroPaginas. Adicione um método exibirDetalhes() que exiba as informações do livro. No método main(), crie dois livros e exiba seus detalhes.

Solução:

```
class Livro:
    def __init__(self, titulo, autor, numero_paginas):
        self.titulo = titulo
        self.autor = autor
        self.numero_paginas = numero_paginas

    # Método para exibir os detalhes do livro
    def exibirDetalhes(self):
        print(f"Título: {self.titulo}")
        print(f"Autor: {self.autor}")
        print(f"Número de páginas: {self.numero_paginas}")
        print()

def main():
    livro1 = Livro("Dom Quixote", "Miguel de Cervantes", 992)
    livro2 = Livro("1984", "George Orwell", 328)
```

```
    livro1.exibirDetalhes()  # Exibindo detalhes do primeiro livro
    livro2.exibirDetalhes()  # Exibindo detalhes do segundo livro

if __name__ == "__main__":
    main()
```

Explicação: A classe Livro tem três atributos: titulo, autor e numero_paginas. O método __init__() é o inicializador da classe, e ele recebe esses valores como parâmetros quando um objeto é criado. O método exibirDetalhes() exibe as informações de um livro: título, autor e número de páginas. No método main(), dois livros são criados com informações distintas e os detalhes de cada um são exibidos. A chamada if __name__ == "__main__": garante que o código no main() seja executado quando o script for rodado diretamente.

Exercício 139: Classe Funcionario

Enunciado: Crie uma classe Funcionario com os atributos nome, salario e cargo. Adicione um método aumentarSalario(double porcentagem) que aumenta o salário de acordo com a porcentagem dada. No método main(), crie um funcionário, aumente seu salário e exiba as informações.

Solução:

```
class Funcionario:
    def __init__(self, nome, salario, cargo):
        self.nome = nome
        self.salario = salario
        self.cargo = cargo

    # Método para aumentar o salário de acordo com a porcentagem
    def aumentarSalario(self, porcentagem):
        self.salario += self.salario * (porcentagem / 100)

    # Método para exibir os detalhes do funcionário
    def exibirDetalhes(self):
        print(f"Nome: {self.nome}")
        print(f"Cargo: {self.cargo}")
        print(f"Salário: R$ {self.salario:.2f}")

def main():
    funcionario = Funcionario("João", 3000.0, "Analista de Sistemas")

    # Aumentando o salário em 10%
    funcionario.aumentarSalario(10)
```

```
    funcionario.exibirDetalhes()  # Exibindo as informações do
funcionário

if __name__ == "__main__":
    main()
```

Explicação:A classe Funcionario tem três atributos: nome, salario e cargo, que são definidos no método __init__().
O método aumentarSalario() recebe a porcentagem de aumento e ajusta o salário do funcionário com base nesse valor.
O método exibirDetalhes() exibe as informações do funcionário, incluindo o nome, cargo e salário formatado para duas casas decimais.
No método main(), criamos um funcionário, aumentamos o salário em 10% e exibimos as informações usando o método exibirDetalhes().

Exercício 140: Classe Fatura

Enunciado: Implemente uma classe Fatura com os atributos numeroItem, descricao, quantidade, precoPorItem. Adicione um método calcularTotal() que retorne o valor total da fatura (quantidade * precoPorItem). No método main(), crie uma fatura e exiba o valor total.

Solução:

```
class Fatura:
    def __init__(self, numero_item, descricao, quantidade,
preco_por_item):
        self.numero_item = numero_item
        self.descricao = descricao
        self.quantidade = quantidade
        self.preco_por_item = preco_por_item

    # Método para calcular o valor total da fatura
    def calcularTotal(self):
        return self.quantidade * self.preco_por_item

def main():
    fatura = Fatura("001", "Notebook", 2, 2500.0)

    total = fatura.calcularTotal()
    print(f"Valor total da fatura: R$ {total:.2f}")
```

```
if __name__ == "__main__":
    main()
```

Explicação: A classe Fatura possui quatro atributos: numero_item, descricao, quantidade e preco_por_item, que são inicializados no método __init__(). O método calcularTotal() retorna o valor total da fatura, multiplicando a quantidade pelo preço por item. No método main(), criamos um objeto Fatura, chamamos o método calcularTotal() e exibimos o valor total formatado com duas casas decimais.

Exercício 141: Classe ContaCorrente

Enunciado: Crie uma classe ContaCorrente com os atributos numeroConta, saldo e limite. Adicione um método verificarLimite() que exiba se o saldo está dentro do limite. No método main(), crie uma conta e teste o método.

Solução:

```
class ContaCorrente:
    def __init__(self, numero_conta, saldo, limite):
        self.numero_conta = numero_conta
        self.saldo = saldo
        self.limite = limite

    # Método para verificar se o saldo está dentro do limite
    def verificarLimite(self):
        if self.saldo > self.limite:
            print("Saldo acima do limite.")
        else:
            print("Saldo dentro do limite.")

def main():
    conta = ContaCorrente(123456, 2000.0, 3000.0)

    conta.verificarLimite()  # Verificando se o saldo está dentro do
limite

if __name__ == "__main__":
    main()
```

Explicação: A classe ContaCorrente possui três atributos: numero_conta, saldo e limite, que são inicializados no método __init__(). O método verificarLimite() compara o saldo com o limite da conta e exibe uma mensagem informando se o saldo está dentro ou acima do

limite. No método main(), criamos um objeto da classe ContaCorrente e chamamos o método verificarLimite() para fazer a verificação.

Exercício 142: Classe Filme

Enunciado: Implemente uma classe Filme com os atributos titulo, diretor e duracao. Adicione um método exibirInformacoes() que exiba as informações do filme. No método main(), crie dois filmes e exiba seus detalhes.

Solução:

```
class Filme:
    def __init__(self, titulo, diretor, duracao):
        self.titulo = titulo
        self.diretor = diretor
        self.duracao = duracao  # duração em minutos

    # Método para exibir as informações do filme
    def exibirInformacoes(self):
        print(f"Título: {self.titulo}")
        print(f"Diretor: {self.diretor}")
        print(f"Duração: {self.duracao} minutos\n")

def main():
    filme1 = Filme("O Poderoso Chefão", "Francis Ford Coppola", 175)
    filme2 = Filme("Pulp Fiction", "Quentin Tarantino", 154)

    filme1.exibirInformacoes()  # Exibindo detalhes do primeiro filme
    filme2.exibirInformacoes()  # Exibindo detalhes do segundo filme

if __name__ == "__main__":
    main()
```

Explicação:A classe Filme possui três atributos: titulo, diretor e duracao, que são inicializados no método __init__(). O método exibirInformacoes() imprime no console as informações sobre o filme. No método main(), criamos dois objetos da classe Filme com títulos, diretores e durações específicos, e chamamos o método exibirInformacoes() para mostrar esses detalhes.

Atributos e Métodos de Instância

Exercício 143: Classe Pessoa com Atributos

Enunciado: Crie uma classe chamada Pessoa com os atributos nome e idade. Adicione um método aniversario() que incremente a idade em 1. No método main(), crie uma pessoa, exiba a idade, chame o método aniversario() e exiba a nova idade.

Solução:

```
class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade

    # Método para incrementar a idade
    def aniversario(self):
        self.idade += 1

def main():
    # Criando uma instância da classe Pessoa
    pessoa = Pessoa("João", 25)

    # Exibindo a idade antes do aniversário
    print(f"Idade antes do aniversário: {pessoa.idade}")

    # Chamando o método aniversario()
    pessoa.aniversario()

    # Exibindo a idade após o aniversário
    print(f"Idade após o aniversário: {pessoa.idade}")

# Chamando o método principal
if __name__ == "__main__":
    main()
```

Explicação: A classe Pessoa tem dois atributos: nome e idade, que são inicializados pelo método __init__. O método aniversario incrementa o valor de idade em 1. A função main cria um objeto da classe Pessoa, exibe a idade inicial, chama o método aniversario e exibe a idade após o incremento. A verificação if __name__ == "__main__": garante que a função main será executada somente se o script for executado diretamente.

Exercício 144: Classe Produto com Desconto

Enunciado: Crie uma classe Produto com os atributos nome, preco e desconto. Adicione um método aplicarDesconto() que aplique o desconto ao preço do produto. No método main(), crie um produto, aplique o desconto e exiba o preço final.

Solução:

```
class Produto:
    def __init__(self, nome, preco, desconto):
        self.nome = nome
        self.preco = preco
        self.desconto = desconto

    # Método para aplicar o desconto ao preço do produto
    def aplicar_desconto(self):
        self.preco -= self.preco * (self.desconto / 100)

def main():
    # Criando uma instância da classe Produto
    produto = Produto("Celular", 2000.0, 10.0)

    # Exibindo o preço original
    print(f"Preço original: R$ {produto.preco:.2f}")

    # Aplicando o desconto
    produto.aplicar_desconto()

    # Exibindo o preço com desconto
    print(f"Preço com desconto: R$ {produto.preco:.2f}")

# Chamando o método principal
if __name__ == "__main__":
    main()
```

Explicação: A classe Produto contém três atributos: nome, preco e desconto, que são inicializados no construtor __init__. O método aplicar_desconto calcula o novo preço do produto aplicando a porcentagem de desconto ao valor atual do preço. A função main cria um objeto da classe Produto, exibe o preço original, chama o método aplicar_desconto e exibe o preço final com desconto. O formato {produto.preco:.2f} garante que o preço seja exibido com duas casas decimais, simulando a exibição monetária.

Exercício 145: Classe Carro com Consumo

Enunciado: Crie uma classe Carro com os atributos marca, modelo, consumoCombustivel (km por litro) e quantidadeCombustivel. Adicione um método dirigir() que reduza o

combustível com base na distância percorrida e no consumo. No método main(), crie um carro, simule uma viagem e exiba o combustível restante.

Solução:

```
class Carro:
    def __init__(self, marca, modelo, consumo_combustivel,
quantidade_combustivel):
        self.marca = marca
        self.modelo = modelo
        self.consumo_combustivel = consumo_combustivel  # Km por litro
        self.quantidade_combustivel = quantidade_combustivel  # Em
litros

    # Método para dirigir o carro, reduzindo o combustível
    def dirigir(self, distancia):
        combustivel_necessario = distancia / self.consumo_combustivel
        if combustivel_necessario <= self.quantidade_combustivel:
            self.quantidade_combustivel -= combustivel_necessario
            print(f"Viagem realizada. Combustível restante:
{self.quantidade_combustivel:.2f} litros.")
        else:
            print("Combustível insuficiente para realizar a viagem.")

def main():
    # Criando uma instância da classe Carro
    carro = Carro("Honda", "Civic", 12.0, 50.0)  # 12 km/l de consumo e
50 litros no tanque

    # Simulando uma viagem de 100 km
    print("Simulando uma viagem de 100 km...")
    carro.dirigir(100)

    # Exibindo o combustível restante
    print(f"Combustível restante no tanque:
{carro.quantidade_combustivel:.2f} litros.")

# Chamando o método principal
if __name__ == "__main__":
    main()
```

Explicação: A classe Carro possui quatro atributos: marca, modelo, consumo_combustivel e quantidade_combustivel. O método dirigir calcula o combustível necessário para uma viagem com base na distância percorrida e no consumo do carro. Se houver combustível suficiente, ele reduz a quantidade disponível no tanque. Caso contrário, exibe uma mensagem informando que o combustível é insuficiente. A função main cria uma instância

da classe Carro, simula uma viagem de 100 km, chama o método dirigir e exibe o combustível restante no tanque. O uso de {self.quantidade_combustivel:.2f} garante a exibição com duas casas decimais para maior precisão e clareza na saída.

Exercício 146: Classe ContaBancaria com Transferência

Enunciado: Crie uma classe ContaBancaria com os atributos saldo e numeroConta. Adicione um método transferir() que receba outra conta bancária e um valor a ser transferido, reduzindo o saldo da conta de origem e aumentando o saldo da conta destino. No método main(), crie duas contas e faça uma transferência.

Solução:

```
class ContaBancaria:
    def __init__(self, numero_conta, saldo):
        self.numero_conta = numero_conta
        self.saldo = saldo

    # Método para transferir dinheiro para outra conta
    def transferir(self, conta_destino, valor):
        if self.saldo >= valor:
            self.saldo -= valor
            conta_destino.saldo += valor
            print(f"Transferência de R$ {valor:.2f} realizada com
sucesso.")
        else:
            print("Saldo insuficiente para a transferência.")

def main():
    # Criando duas contas bancárias
    conta1 = ContaBancaria(12345, 1000.0)
    conta2 = ContaBancaria(67890, 500.0)

    # Exibindo o saldo inicial das contas
    print(f"Saldo da Conta 1: R$ {conta1.saldo:.2f}")
    print(f"Saldo da Conta 2: R$ {conta2.saldo:.2f}")

    # Realizando uma transferência de R$ 200 da conta 1 para a conta 2
    print("\nRealizando transferência de R$ 200.00 da Conta 1 para a
Conta 2...")
    conta1.transferir(conta2, 200.0)

    # Exibindo o saldo final das contas
    print(f"\nSaldo da Conta 1 após transferência: R$
```

```
{conta1.saldo:.2f}")
    print(f"Saldo da Conta 2 após transferência: R$ {conta2.saldo:.2f}")

# Chamando o método principal
if __name__ == "__main__":
    main()
```

Explicação: A classe ContaBancaria possui dois atributos: numero_conta e saldo, que são inicializados no construtor __init__. O método transferir recebe como parâmetros outra conta (conta_destino) e o valor a ser transferido. Ele verifica se o saldo da conta de origem é suficiente antes de realizar a transferência. A função main cria duas instâncias de ContaBancaria, exibe os saldos iniciais, realiza uma transferência de R$ 200,00 da Conta 1 para a Conta 2 e, em seguida, exibe os saldos finais. O uso de {:.2f} garante que os valores monetários sejam exibidos com duas casas decimais.

Exercício 147: Classe Funcionario com Bônus

Enunciado: Implemente uma classe Funcionario com os atributos nome e salario. Adicione um método receberBonus(double percentual) que aumente o salário com base em um percentual. No método main(), crie um funcionário, aplique o bônus e exiba o novo salário.

Solução:

```
class Funcionario:
    def __init__(self, nome, salario):
        self.nome = nome
        self.salario = salario

    # Método para aplicar o bônus no salário
    def receber_bonus(self, percentual):
        self.salario += self.salario * (percentual / 100)

def main():
    # Criando um funcionário
    funcionario = Funcionario("Mariana", 5000.0)

    # Exibindo o salário antes do bônus
    print(f"Salário antes do bônus: R$ {funcionario.salario:.2f}")

    # Aplicando um bônus de 10%
    funcionario.receber_bonus(10)
```

```
    # Exibindo o salário após o bônus
    print(f"Salário após o bônus: R$ {funcionario.salario:.2f}")

# Chamando o método principal
if __name__ == "__main__":
    main()
```

Explicação: A classe Funcionario possui dois atributos: nome e salario, que são inicializados no método construtor __init__. O método receber_bonus aceita um percentual como parâmetro e calcula o aumento no salário com base nesse percentual. A função main cria uma instância da classe Funcionario com o nome "Mariana" e salário inicial de R$ 5000, exibe o salário inicial, aplica um bônus de 10% e exibe o salário final. O uso de {:.2f} garante que os valores sejam formatados com duas casas decimais para manter a precisão monetária.

Exercício 148: Classe Aluno com Aprovação

Enunciado: Crie uma classe Aluno com os atributos nome, nota1 e nota2. Adicione um método verificarAprovacao() que calcule a média e retorne se o aluno foi aprovado (média maior ou igual a 7). No método main(), crie um aluno e verifique sua aprovação.

Solução:

```
class Aluno:
    def __init__(self, nome, nota1, nota2):
        self.nome = nome
        self.nota1 = nota1
        self.nota2 = nota2

    # Método para verificar se o aluno foi aprovado
    def verificar_aprovacao(self):
        media = (self.nota1 + self.nota2) / 2
        return media >= 7.0

def main():
    # Criando um aluno
    aluno = Aluno("Lucas", 8.0, 6.5)

    # Verificando a aprovação
    if aluno.verificar_aprovacao():
        print(f"{aluno.nome} foi aprovado.")
```

```
    else:
        print(f"{aluno.nome} foi reprovado.")

# Chamando o método principal
if __name__ == "__main__":
    main()
```

Explicação: A classe Aluno possui os atributos nome, nota1 e nota2, inicializados no método construtor __init__. O método verificar_aprovacao calcula a média das duas notas e retorna True se a média for maior ou igual a 7, ou False caso contrário. Na função main, criamos um objeto da classe Aluno com o nome "Lucas" e suas notas. Verificamos se o aluno foi aprovado chamando o método verificar_aprovacao. A mensagem correspondente é exibida com base no resultado.

Exercício 149: Classe Retângulo com Alteração de Dimensões

Enunciado: Crie uma classe Retangulo com os atributos largura e altura. Adicione métodos alterarDimensoes(double novaLargura, double novaAltura) e calcularArea() para atualizar as dimensões e calcular a área. No método main(), altere as dimensões de um retângulo e exiba a nova área.

Solução:

```
class Retangulo:
    def __init__(self, largura, altura):
        self.largura = largura
        self.altura = altura

    # Método para alterar as dimensões do retângulo
    def alterar_dimensoes(self, nova_largura, nova_altura):
        self.largura = nova_largura
        self.altura = nova_altura

    # Método para calcular a área do retângulo
    def calcular_area(self):
        return self.largura * self.altura

def main():
    # Criando um retângulo com dimensões iniciais
    retangulo = Retangulo(5.0, 3.0)
```

```
    # Exibindo a área inicial
    print(f"Área inicial: {retangulo.calcular_area()}")

    # Alterando as dimensões
    retangulo.alterar_dimensoes(10.0, 8.0)

    # Exibindo a nova área
    print(f"Nova área: {retangulo.calcular_area()}")

# Chamando o método principal
if __name__ == "__main__":
    main()
```

Explicação: Construtor __init__: Inicializa os atributos largura e altura. Método alterar_dimensoes: Atualiza os valores de largura e altura com as novas dimensões fornecidas. Método calcular_area: Retorna o valor da área calculado como o produto de largura e altura.
Função main:
Cria uma instância de Retangulo com dimensões iniciais. Calcula e exibe a área inicial usando o método calcular_area. Altera as dimensões do retângulo usando alterar_dimensoes. Calcula e exibe a nova área após a alteração.

Exercício 150: Classe Produto com Aumento de Preço

Enunciado: Implemente uma classe Produto com os atributos nome, preco e quantidade. Adicione um método aumentarPreco(double porcentagem) que aumente o preço com base na porcentagem. No método main(), crie um produto, aumente o preço e exiba o novo valor.

Solução:

```
class Produto:
    def __init__(self, nome, preco, quantidade):
        self.nome = nome
        self.preco = preco
        self.quantidade = quantidade

    # Método para aumentar o preço com base na porcentagem
    def aumentar_preco(self, porcentagem):
        self.preco += self.preco * (porcentagem / 100)

def main():
    # Criando um produto
```

```
    produto = Produto("Geladeira", 1500.0, 20)

    # Exibindo o preço inicial
    print(f"Preço inicial: R$ {produto.preco:.2f}")

    # Aumentando o preço em 15%
    produto.aumentar_preco(15)

    # Exibindo o novo preço
    print(f"Preço após aumento: R$ {produto.preco:.2f}")

# Chamando o método principal
if __name__ == "__main__":
    main()
```

Explicação: Construtor __init__: Inicializa os atributos nome, preco e quantidade. Método aumentar_preco: Aplica o aumento percentual ao preço do produto usando a fórmula preco += preco * (porcentagem / 100).
Função main:
Cria uma instância da classe Produto com valores iniciais. Exibe o preço inicial do produto formatado com duas casas decimais. Aumenta o preço em 15% utilizando o método aumentar_preco. Exibe o novo preço após o aumento.

Exercício 151: Classe Pessoa com Altura e Peso

Enunciado: Crie uma classe Pessoa com os atributos nome, altura e peso. Adicione métodos alterarPeso(double novoPeso) e alterarAltura(double novaAltura). No método main(), altere o peso e a altura de uma pessoa e exiba os valores atualizados.

Solução:

```
class Pessoa:
    def __init__(self, nome, altura, peso):
        self.nome = nome
        self.altura = altura
        self.peso = peso

    # Método para alterar o peso da pessoa
    def alterar_peso(self, novo_peso):
        self.peso = novo_peso

    # Método para alterar a altura da pessoa
```

```
    def alterar_altura(self, nova_altura):
        self.altura = nova_altura

def main():
    # Criando uma pessoa
    pessoa = Pessoa("Joana", 1.70, 65.0)

    # Exibindo os valores iniciais
    print(f"Altura inicial: {pessoa.altura:.2f} m")
    print(f"Peso inicial: {pessoa.peso:.2f} kg")

    # Alterando peso e altura
    pessoa.alterar_peso(68.0)
    pessoa.alterar_altura(1.72)

    # Exibindo os novos valores
    print(f"Nova altura: {pessoa.altura:.2f} m")
    print(f"Novo peso: {pessoa.peso:.2f} kg")

# Chamando o método principal
if __name__ == "__main__":
    main()
```

Explicação: Construtor __init__: Inicializa os atributos nome, altura e peso.
Métodos alterar_peso e alterar_altura: alterar_peso: Atualiza o valor do atributo peso com o novo peso fornecido. alterar_altura: Atualiza o valor do atributo altura com a nova altura fornecida.
Função main:
Cria uma instância da classe Pessoa com valores iniciais. Exibe os valores iniciais de altura e peso formatados com duas casas decimais. Usa os métodos alterar_peso e alterar_altura para atualizar os valores. Exibe os valores atualizados formatados.

Exercício 152: Classe Banco com Saldo Total

Enunciado: Crie uma classe Banco que tenha uma lista de contas bancárias como atributo. Adicione um método calcularSaldoTotal() que some os saldos de todas as contas. No método main(), crie várias contas, adicione-as ao banco e exiba o saldo total.

Solução:

```
class ContaBancaria:
    def __init__(self, numero_conta, saldo):
        self.numero_conta = numero_conta
```

```
        self.saldo = saldo


class Banco:
    def __init__(self):
        self.contas = []

    # Método para calcular o saldo total do banco
    def calcular_saldo_total(self):
        saldo_total = sum(conta.saldo for conta in self.contas)
        return saldo_total


def main():
    banco = Banco()

    # Criando contas bancárias
    conta1 = ContaBancaria(12345, 1000.0)
    conta2 = ContaBancaria(67890, 2000.0)

    # Adicionando as contas ao banco
    banco.contas.append(conta1)
    banco.contas.append(conta2)

    # Exibindo o saldo total do banco
    print(f"Saldo total no banco: R$
{banco.calcular_saldo_total():.2f}")


# Chamando o método principal
if __name__ == "__main__":
    main()
```

Explicação: Classe ContaBancaria: Representa uma conta bancária com os atributos numero_conta e saldo.O método __init__ inicializa esses atributos ao criar uma conta.
Classe Banco: Contém o atributo contas, que é uma lista de objetos ContaBancaria.
O método calcular_saldo_total percorre a lista de contas usando uma compreensão de lista e soma os saldos de todas as contas.
Função main:
Cria um objeto Banco.
Instancia duas contas bancárias com número de conta e saldo.
Adiciona as contas à lista de contas do banco.
Calcula e exibe o saldo total do banco formatado com duas casas decimais.

Exercício 153: Classe Livro com Empréstimo

Enunciado: Implemente uma classe Livro com os atributos titulo, autor e disponivel. Adicione um método emprestar() que marque o livro como indisponível e outro método devolver() que marque como disponível. No método main(), crie um livro e simule um empréstimo e uma devolução.

Solução:

```
class Livro:
    def __init__(self, titulo, autor):
        self.titulo = titulo
        self.autor = autor
        self.disponivel = True

    # Método para emprestar o livro
    def emprestar(self):
        if self.disponivel:
            self.disponivel = False
            print("O livro foi emprestado.")
        else:
            print("O livro já está emprestado.")

    # Método para devolver o livro
    def devolver(self):
        if not self.disponivel:
            self.disponivel = True
            print("O livro foi devolvido.")
        else:
            print("O livro já estava disponível.")


def main():
    livro = Livro("1984", "George Orwell")

    # Simulando o empréstimo e a devolução do livro
    livro.emprestar()
    livro.devolver()


# Chamando o método principal
if __name__ == "__main__":
    main()
```

Explicação: Classe Livro:

Atributos titulo, autor, e disponivel. O atributo disponivel é iniciado como True, indicando que o livro está disponível no momento.
O método emprestar verifica se o livro está disponível (disponivel == True). Se estiver, o livro é marcado como emprestado (disponivel = False). Caso contrário, uma mensagem informando que o livro já está emprestado é exibida.
O método devolver verifica se o livro foi emprestado (disponivel == False). Se sim, o livro é marcado como disponível novamente (disponivel = True). Caso contrário, uma mensagem indicando que o livro já estava disponível é exibida.
Função main:
Cria um objeto Livro com o título "1984" e o autor "George Orwell".
Simula o processo de empréstimo e devolução do livro, chamando os métodos emprestar e devolver.

Exercício 154: Classe Veículo com Abastecimento

Enunciado: Crie uma classe Veiculo com os atributos modelo e combustivel (em litros). Adicione um método abastecer(double litros) que aumente a quantidade de combustível. No método main(), crie um veículo, simule um abastecimento e exiba o combustível atualizado.

Solução:

```
class Veiculo:
    def __init__(self, modelo, combustivel):
        self.modelo = modelo
        self.combustivel = combustivel

    # Método para abastecer o veículo
    def abastecer(self, litros):
        self.combustivel += litros


def main():
    veiculo = Veiculo("SUV", 20.0)

    # Exibindo o combustível inicial
    print(f"Combustível inicial: {veiculo.combustivel} litros")

    # Abastecendo o veículo com 30 litros
    veiculo.abastecer(30)

    # Exibindo o combustível após o abastecimento
    print(f"Combustível após abastecimento: {veiculo.combustivel}
litros")
```

```
# Chamando o método principal
if __name__ == "__main__":
    main()
```

Explicação: Classe Veiculo:
Atributos modelo e combustivel. O atributo modelo define o tipo de veículo (como "SUV"), enquanto combustivel armazena a quantidade de combustível em litros. O método abastecer recebe um número de litros e adiciona esse valor ao combustível atual do veículo.
Função main:
Cria um objeto Veiculo com o modelo "SUV" e uma quantidade inicial de combustível de 20 litros. Exibe o combustível antes e depois de abastecer o veículo com 30 litros.

Exercício 155: Classe Loja com Produtos

Enunciado: Crie uma classe Loja com o atributo nome e uma lista de produtos. Adicione um método adicionarProduto(Produto produto) que adicione um produto à lista e um método exibirProdutos() que exiba todos os produtos. No método main(), crie uma loja, adicione produtos e exiba a lista.

Solução:

```
class Produto:
    def __init__(self, nome, preco):
        self.nome = nome
        self.preco = preco


class Loja:
    def __init__(self, nome):
        self.nome = nome
        self.produtos = []

    # Método para adicionar produto à lista
    def adicionar_produto(self, produto):
        self.produtos.append(produto)

    # Método para exibir todos os produtos
    def exibir_produtos(self):
        print(f"Produtos da loja {self.nome}:")
        for produto in self.produtos:
            print(f"{produto.nome} - R$ {produto.preco}")
```

```
def main():
    loja = Loja("Loja de Eletrônicos")

    # Criando produtos
    produto1 = Produto("Smartphone", 1500.0)
    produto2 = Produto("Televisão", 3000.0)

    # Adicionando produtos à loja
    loja.adicionar_produto(produto1)
    loja.adicionar_produto(produto2)

    # Exibindo os produtos da loja
    loja.exibir_produtos()


# Chamando o método principal
if __name__ == "__main__":
    main()
```

Explicação: Classe Produto:
Atributos nome e preco são usados para definir as características de um produto.
Classe Loja: Atributo nome para armazenar o nome da loja e produtos que é uma lista para armazenar os objetos Produto. O método adicionar_produto() adiciona um produto à lista de produtos. O método exibir_produtos() percorre a lista de produtos e imprime o nome e o preço de cada um.
Função main(): Cria uma loja e dois produtos. Adiciona os produtos à loja e exibe a lista de produtos.

Exercício 156: Classe Cesta de Compras com Total

Enunciado: Crie uma classe CestaDeCompras com uma lista de produtos. Adicione um método calcularTotal() que some o valor total dos produtos da cesta. No método main(), adicione produtos à cesta e exiba o valor total.

Solução:

```
class Produto:
    def __init__(self, nome, preco):
        self.nome = nome
        self.preco = preco
```

```
class CestaDeCompras:
    def __init__(self):
        self.produtos = []

    # Método para calcular o valor total da cesta de compras
    def calcular_total(self):
        total = 0
        for produto in self.produtos:
            total += produto.preco
        return total


def main():
    cesta = CestaDeCompras()

    # Criando produtos
    produto1 = Produto("Notebook", 3000.0)
    produto2 = Produto("Fone de Ouvido", 200.0)

    # Adicionando produtos à cesta
    cesta.produtos.append(produto1)
    cesta.produtos.append(produto2)

    # Exibindo o valor total da cesta
    print(f"Valor total da cesta: R$ {cesta.calcular_total()}")


# Chamando o método principal
if __name__ == "__main__":
    main()
```

Explicação: Classe Produto: Contém os atributos nome e preco que definem os produtos.
Classe CestaDeCompras: A lista produtos armazena objetos do tipo Produto.
O método calcular_total() percorre a lista de produtos e soma os preços de cada item, retornando o valor total.
Função main():Cria uma cesta de compras e dois produtos. Adiciona os produtos à cesta e exibe o valor total calculado.

Exercício 157: Classe Restaurante com Pedidos

Enunciado: Implemente uma classe Restaurante com uma lista de pedidos. Adicione um método adicionarPedido(String pedido) e outro método exibirPedidos() para exibir todos os pedidos. No método main(), crie um restaurante, adicione pedidos e exiba a lista.

Solução:

```
class Restaurante:
    def __init__(self):
        self.pedidos = []

    # Método para adicionar um pedido
    def adicionar_pedido(self, pedido):
        self.pedidos.append(pedido)

    # Método para exibir todos os pedidos
    def exibir_pedidos(self):
        print("Pedidos realizados:")
        for pedido in self.pedidos:
            print(pedido)


def main():
    restaurante = Restaurante()

    # Adicionando pedidos
    restaurante.adicionar_pedido("Pizza")
    restaurante.adicionar_pedido("Lasanha")
    restaurante.adicionar_pedido("Hambúrguer")

    # Exibindo os pedidos
    restaurante.exibir_pedidos()


# Chamando o método principal
if __name__ == "__main__":
    main()
```

Explicação: Classe Restaurante: A lista pedidos armazena os pedidos feitos, que são representados como strings. O método adicionar_pedido() adiciona um novo pedido à lista. O método exibir_pedidos() percorre a lista de pedidos e os exibe.
Função main(): Cria um restaurante, adiciona alguns pedidos e exibe todos os pedidos na lista.

Encapsulamento (getters e setters)

Exercício 158: Classe ContaBancaria com Encapsulamento

Enunciado: Implemente uma classe ContaBancaria com os atributos saldo e numeroConta, encapsulados com os métodos getSaldo(), setSaldo(), getNumeroConta() e setNumeroConta(). No método main(), crie uma conta bancária, defina seus valores e exiba o saldo usando os getters.

Solução:

```
class ContaBancaria:
    def __init__(self):
        self._saldo = 0.0
        self._numeroConta = 0

    # Getter para saldo
    def get_saldo(self):
        return self._saldo

    # Setter para saldo
    def set_saldo(self, saldo):
        if saldo >= 0:
            self._saldo = saldo
        else:
            print("Saldo não pode ser negativo.")

    # Getter para número da conta
    def get_numero_conta(self):
        return self._numeroConta

    # Setter para número da conta
    def set_numero_conta(self, numeroConta):
        self._numeroConta = numeroConta


def main():
    conta = ContaBancaria()

    # Definindo valores através dos setters
    conta.set_numero_conta(12345)
    conta.set_saldo(1000.0)

    # Exibindo valores através dos getters
    print(f"Número da conta: {conta.get_numero_conta()}")
    print(f"Saldo da conta: R$ {conta.get_saldo()}")
```

```
# Chamando o método principal
if __name__ == "__main__":
    main()
```

Explicação: Classe ContaBancaria: Os atributos saldo e numeroConta são privados (utilizando o prefixo _ por convenção), garantindo o encapsulamento. Os métodos get_saldo() e set_saldo() permitem acessar e modificar o saldo, respectivamente. O setter valida que o saldo não seja negativo. Os métodos get_numero_conta() e set_numero_conta() permitem acessar e modificar o número da conta.
Função main():
Cria uma instância de ContaBancaria, define o número da conta e o saldo utilizando os setters, e exibe os valores através dos getters.

Exercício 159: Classe Produto com Validação de Preço

Enunciado: Crie uma classe Produto com os atributos nome e preco. O atributo preco deve ser encapsulado, e o método setPreco() deve garantir que o preço não seja negativo. No método main(), crie um produto, tente definir um preço negativo e exiba a mensagem de erro.

Solução:

```
class Produto:
    def __init__(self, nome=""):
        self.nome = nome
        self._preco = 0.0

    # Getter para o preço
    def get_preco(self):
        return self._preco

    # Setter para o preço com validação
    def set_preco(self, preco):
        if preco >= 0:
            self._preco = preco
        else:
            print("Erro: O preço não pode ser negativo.")

    # Getter e Setter para o nome
    def get_nome(self):
        return self.nome

    def set_nome(self, nome):
        self.nome = nome
```

```
def main():
    produto = Produto()

    # Definindo nome e tentando definir um preço negativo
    produto.set_nome("Smartphone")
    produto.set_preco(-500.0)  # Deverá exibir uma mensagem de erro

    # Definindo um preço válido
    produto.set_preco(2000.0)

    # Exibindo o nome e preço do produto
    print(f"Produto: {produto.get_nome()}")
    print(f"Preço: R$ {produto.get_preco()}")


# Chamando o método principal
if __name__ == "__main__":
    main()
```

Explicação: Classe Produto:O atributo preco é encapsulado utilizando o prefixo _, seguindo as convenções de Python para atributos privados. O método set_preco() valida se o preço não é negativo. Se for, imprime uma mensagem de erro.O método get_preco() retorna o valor do preço, e o método get_nome() retorna o nome do produto.
Função main(): Cria um objeto Produto, define o nome e tenta definir um preço negativo. Isso irá disparar a mensagem de erro. Depois, define um preço válido e exibe o nome e o preço usando os getters.

Exercício 160: Classe Pessoa com Validação de Idade

Enunciado: Implemente uma classe Pessoa com os atributos nome e idade. Encapsule a idade e adicione uma validação no método setIdade() para garantir que a idade seja maior que 0. No método main(), crie uma pessoa, defina uma idade inválida e exiba a mensagem de erro.

Solução:

```
class Pessoa:
    def __init__(self, nome=""):
        self.nome = nome
        self._idade = 0

    # Getter para idade
```

```
    def get_idade(self):
        return self._idade

    # Setter para idade com validação
    def set_idade(self, idade):
        if idade > 0:
            self._idade = idade
        else:
            print("Erro: A idade deve ser maior que 0.")

    # Getter e Setter para nome
    def get_nome(self):
        return self.nome

    def set_nome(self, nome):
        self.nome = nome


def main():
    pessoa = Pessoa()

    # Definindo nome e uma idade inválida
    pessoa.set_nome("Carlos")
    pessoa.set_idade(-5)  # Deverá exibir uma mensagem de erro

    # Definindo uma idade válida
    pessoa.set_idade(25)

    # Exibindo os dados da pessoa
    print(f"Nome: {pessoa.get_nome()}")
    print(f"Idade: {pessoa.get_idade()}")


# Chamando o método principal
if __name__ == "__main__":
    main()
```

Explicação: Classe Pessoa: O atributo idade é encapsulado utilizando o prefixo _, seguindo as convenções de Python para atributos privados. O método set_idade() valida se a idade fornecida é maior que 0. Caso contrário, exibe uma mensagem de erro. O método get_idade() retorna o valor da idade, e o método get_nome() retorna o nome da pessoa. Função main(): Cria um objeto Pessoa, define o nome e tenta definir uma idade inválida, o que resultará na exibição de uma mensagem de erro.
Depois, define uma idade válida e exibe o nome e a idade usando os getters.

Exercício 161: Classe Funcionario com Bônus Encapsulado

Enunciado: Crie uma classe Funcionario com os atributos nome, salario e bonus. O atributo bonus deve ser calculado como uma porcentagem do salário e encapsulado com getBonus() e setBonus(). No método main(), defina o salário, calcule o bônus e exiba os valores usando os getters.

Solução:

```
class Funcionario:
    def __init__(self, nome="", salario=0.0):
        self.nome = nome
        self.salario = salario
        self._bonus = 0.0

    # Getter para o salário
    def get_salario(self):
        return self.salario

    # Setter para o salário
    def set_salario(self, salario):
        self.salario = salario

    # Getter para o bônus
    def get_bonus(self):
        return self._bonus

    # Setter para o bônus (calculado como uma porcentagem do salário)
    def set_bonus(self, percentual):
        self._bonus = self.salario * (percentual / 100)

    # Getter e Setter para o nome
    def get_nome(self):
        return self.nome

    def set_nome(self, nome):
        self.nome = nome


def main():
    funcionario = Funcionario()

    # Definindo o nome e salário
    funcionario.set_nome("Ana")
    funcionario.set_salario(5000.0)
```

```
    # Definindo o bônus como 10% do salário
    funcionario.set_bonus(10)

    # Exibindo os dados do funcionário
    print(f"Funcionário: {funcionario.get_nome()}")
    print(f"Salário: R$ {funcionario.get_salario()}")
    print(f"Bônus: R$ {funcionario.get_bonus()}")


# Chamando o método principal
if __name__ == "__main__":
    main()
```

Explicação: Classe Funcionario: O atributo salario é definido e pode ser alterado diretamente com o setter set_salario(). O atributo _bonus é encapsulado e calculado com base no salário através do método set_bonus(), que calcula o bônus como uma porcentagem do salário. O getter get_bonus() retorna o valor do bônus, enquanto o getter get_salario() retorna o salário. O nome do funcionário é armazenado e acessado através de get_nome() e set_nome().
Função main(): Cria um objeto Funcionario, define o nome e o salário.
O bônus é calculado como 10% do salário utilizando o setter set_bonus().
O nome, salário e bônus do funcionário são exibidos usando os getters.

Exercício 162: Classe Veiculo com Encapsulamento de Combustível

Enunciado: Implemente uma classe Veiculo com o atributo combustivel. O combustível deve ser encapsulado, e o método setCombustivel() deve garantir que o valor esteja entre 0 e o valor máximo de combustível do tanque. No método main(), tente definir um valor maior que o permitido e exiba a mensagem de erro.

Solução:

```
class Veiculo:
    def __init__(self):
        self._combustivel = 0.0
        self.capacidade_tanque = 50.0  # Capacidade máxima de
combustível

    # Getter para combustível
    def get_combustivel(self):
        return self._combustivel
```

```
    # Setter para combustível com validação
    def set_combustivel(self, combustivel):
        if 0 <= combustivel <= self.capacidade_tanque:
            self._combustivel = combustivel
        else:
            print(f"Erro: Combustível deve estar entre 0 e
{self.capacidade_tanque} litros.")

def main():
    veiculo = Veiculo()

    # Tentando definir um valor de combustível inválido
    veiculo.set_combustivel(60.0)  # Deve exibir uma mensagem de erro

    # Definindo um valor válido
    veiculo.set_combustivel(30.0)

    # Exibindo o valor de combustível
    print(f"Combustível no tanque: {veiculo.get_combustivel()} litros.")

# Chamando o método principal
if __name__ == "__main__":
    main()
```

Explicação: Classe Veiculo: O atributo _combustivel é encapsulado. A capacidade máxima do tanque (capacidade_tanque) é definida como 50.0 litros. O método set_combustivel() garante que o valor de combustível esteja dentro do intervalo válido (0 a 50 litros). Se o valor fornecido for inválido, uma mensagem de erro é exibida. O getter get_combustivel() retorna o valor atual do combustível.
Função main(): Cria um objeto Veiculo e tenta definir um valor de combustível inválido (60.0 litros), o que aciona a mensagem de erro. Em seguida, define um valor válido (30.0 litros) e exibe o combustível no tanque.

Conclusão do Capítulo 7: Classes e Objetos

Neste capítulo, exploramos os conceitos fundamentais da Programação Orientada a Objetos (POO), focando na criação de classes e objetos, definição de atributos e métodos, e a prática do encapsulamento por meio de getters e setters. Através de exercícios práticos, foi possível compreender como organizar o código em torno de objetos, permitindo uma estrutura mais modular, reutilizável e fácil de manter.

Primeiramente, aprendemos a definir classes e objetos, que são a base da POO. Os exercícios demonstraram como criar instâncias de classes, manipular atributos e métodos, e utilizar esses objetos para representar entidades reais, como carros, pessoas e produtos.

Em seguida, abordamos o uso de atributos e métodos de instância, mostrando como encapsular informações e definir o comportamento dos objetos. Esse conceito permitiu explorar a capacidade de alterar dinamicamente os atributos, utilizando métodos específicos para lidar com as interações entre os objetos.

Por fim, vimos como aplicar o encapsulamento com getters e setters, protegendo os dados internos das classes e garantindo que somente valores válidos sejam atribuídos aos atributos. O encapsulamento é essencial para garantir a integridade dos dados e proporcionar um controle maior sobre as modificações internas dos objetos.

Introdução ao Capítulo 8: Herança e Polimorfismo

No Capítulo 8, vamos explorar dois dos pilares fundamentais da Programação Orientada a Objetos (POO): Herança e Polimorfismo. Esses conceitos permitem a criação de sistemas mais flexíveis, reutilizáveis e eficientes.

Primeiramente, veremos a herança, que possibilita que uma classe derive características de outra, permitindo o reaproveitamento de código e a criação de hierarquias entre classes. A sobrescrita de métodos permitirá que classes filhas modifiquem o comportamento de métodos herdados da classe mãe.

Por fim, vamos aprender sobre o polimorfismo, que permite que objetos de diferentes classes sejam tratados de maneira uniforme, tornando o código mais dinâmico e adaptável. Este capítulo mostrará como esses conceitos se complementam para melhorar a estrutura e flexibilidade de sistemas orientados a objetos.

Herança Simples

Exercício 163: Classe Animal e Subclasse Cachorro

Enunciado: Crie uma classe Animal com os atributos nome e idade. Em seguida, crie uma subclasse Cachorro que herda de Animal e tenha o método latir(), que exibe uma mensagem no console. No método main(), crie um objeto Cachorro e chame seus métodos.

Código de Solução:

```
class Animal:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade
```

```
    def exibir_detalhes(self):
        print(f"Nome: {self.nome}")
        print(f"Idade: {self.idade} anos")

class Cachorro(Animal):
    def latir(self):
        print("O cachorro está latindo: Au Au!")

def main():
    cachorro = Cachorro("Rex", 3)

    cachorro.exibir_detalhes()
    cachorro.latir()

if __name__ == "__main__":
    main()
```

Explicação: Classe Animal: Define os atributos nome e idade, e o método exibir_detalhes() para exibir essas informações.
Classe Cachorro: Herda de Animal e adiciona o método latir(), que exibe uma mensagem específica para um cachorro.
Função main(): Cria uma instância da classe Cachorro, define seu nome e idade (herdados da classe Animal), e chama os métodos exibir_detalhes() e latir().

Exercício 164: Classe Veiculo e Subclasse Carro

Enunciado: Crie uma classe Veiculo com os atributos marca e modelo. Crie uma subclasse Carro que herda de Veiculo e tenha um atributo adicional ano. No método main(), crie um objeto Carro e exiba todos os atributos herdados e o atributo ano.

Código de Solução:

```
class Veiculo:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

    def exibir_detalhes(self):
        print(f"Marca: {self.marca}")
        print(f"Modelo: {self.modelo}")

class Carro(Veiculo):
```

```
    def __init__(self, marca, modelo, ano):
        super().__init__(marca, modelo)  # Chama o construtor da classe
mãe (Veiculo)
        self.ano = ano

    def exibir_ano(self):
        print(f"Ano: {self.ano}")

def main():
    carro = Carro("Toyota", "Corolla", 2020)

    carro.exibir_detalhes()
    carro.exibir_ano()

if __name__ == "__main__":
    main()
```

Explicação: Classe Veiculo: Define os atributos marca e modelo, além do método exibir_detalhes() que exibe essas informações.
Classe Carro: Herda de Veiculo e adiciona o atributo ano e o método exibir_ano() para mostrar o ano do carro.
Função main(): Cria uma instância de Carro, define os valores para os atributos herdados de Veiculo e o atributo ano, e chama os métodos exibir_detalhes() e exibir_ano().

Exercício 165: Classe Pessoa e Subclasse Funcionario

Enunciado: Implemente uma classe Pessoa com os atributos nome e idade. Crie uma subclasse Funcionario que herda de Pessoa e tenha o atributo salario. No método main(), crie um objeto Funcionario, atribua valores aos atributos e exiba as informações no console.

Código de Solução:

```
class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade

    def exibir_informacoes(self):
        print(f"Nome: {self.nome}")
        print(f"Idade: {self.idade} anos")

class Funcionario(Pessoa):
```

```
    def __init__(self, nome, idade, salario):
        super().__init__(nome, idade)  # Chama o construtor da classe
mãe (Pessoa)
        self.salario = salario

    def exibir_salario(self):
        print(f"Salário: R$ {self.salario:.2f}")

def main():
    funcionario = Funcionario("Maria", 30, 4000.0)

    funcionario.exibir_informacoes()
    funcionario.exibir_salario()

if __name__ == "__main__":
    main()
```

Explicação: Classe Pessoa: Define os atributos nome e idade, além do método exibir_informacoes(), que exibe essas informações.
Classe Funcionario: Herda os atributos e métodos de Pessoa, adicionando o atributo salario e o método exibir_salario() para exibir o salário.
Função main(): Cria uma instância de Funcionario, define os valores para os atributos herdados de Pessoa e o novo atributo salario, e chama os métodos exibir_informacoes() e exibir_salario().

Exercício 166: Classe Conta e Subclasse ContaCorrente

Enunciado: Crie uma classe Conta com o atributo saldo e um método depositar(). Em seguida, crie uma subclasse ContaCorrente que herda de Conta e adicione um método sacar(). No método main(), crie um objeto ContaCorrente, realize um depósito e um saque, e exiba o saldo final.

Código de Solução:

```
class Conta:
    def __init__(self):
        self.saldo = 0.0

    def depositar(self, valor):
        self.saldo += valor
        print(f"Depósito de R$ {valor:.2f} realizado. Saldo atual: R$
{self.saldo:.2f}")
```

```
class ContaCorrente(Conta):
    def sacar(self, valor):
        if valor <= self.saldo:
            self.saldo -= valor
            print(f"Saque de R$ {valor:.2f} realizado. Saldo atual: R$
{self.saldo:.2f}")
        else:
            print("Saldo insuficiente para realizar o saque.")

def main():
    conta = ContaCorrente()
    conta.depositar(1000.0)
    conta.sacar(500.0)
    conta.sacar(600.0)  # Tentativa de saque maior que o saldo

if __name__ == "__main__":
    main()
```

Explicação: Classe Conta: Possui o atributo saldo e o método depositar(), que aumenta o saldo.
Classe ContaCorrente: Herda de Conta e adiciona o método sacar(), que permite o saque desde que o saldo seja suficiente.
Função main(): Cria uma instância de ContaCorrente, realiza um depósito e dois saques (um válido e outro com saldo insuficiente).

Exercício 167: Classe Produto e Subclasse Livro

Enunciado: Crie uma classe Produto com os atributos nome e preco. Crie uma subclasse Livro que herda de Produto e tenha o atributo autor. No método main(), crie um objeto Livro, atribua valores a seus atributos e exiba todas as informações.

Código de Solução:

```
class Produto:
    def __init__(self, nome, preco):
        self.nome = nome
        self.preco = preco

    def exibir_detalhes(self):
        print(f"Nome: {self.nome}")
        print(f"Preço: R$ {self.preco:.2f}")
```

```
class Livro(Produto):
    def __init__(self, nome, preco, autor):
        super().__init__(nome, preco)
        self.autor = autor

    def exibir_autor(self):
        print(f"Autor: {self.autor}")

def main():
    livro = Livro("O Senhor dos Anéis", 59.90, "J.R.R. Tolkien")
    livro.exibir_detalhes()
    livro.exibir_autor()

if __name__ == "__main__":
    main()
```

Explicação: Classe Produto: Define os atributos nome e preco e o método exibir_detalhes() para exibir essas informações.
Classe Livro: Herda de Produto e adiciona o atributo autor e o método exibir_autor() para exibir o autor.
Função main(): Cria uma instância de Livro, define os valores para todos os atributos e exibe as informações.

Exercício 168: Classe Pessoa e Subclasse Aluno

Enunciado: Implemente uma classe Pessoa com os atributos nome e cpf. Crie uma subclasse Aluno que herda de Pessoa e adicione o atributo matricula. No método main(), crie um objeto Aluno e exiba os dados herdados e o número de matrícula.

Código de Solução:

```
class Pessoa:
    def __init__(self, nome, cpf):
        self.nome = nome
        self.cpf = cpf

    def exibir_informacoes(self):
        print(f"Nome: {self.nome}")
        print(f"CPF: {self.cpf}")

class Aluno(Pessoa):
    def __init__(self, nome, cpf, matricula):
        super().__init__(nome, cpf)
        self.matricula = matricula
```

```
    def exibir_matricula(self):
        print(f"Matrícula: {self.matricula}")

def main():
    aluno = Aluno("João", "123.456.789-00", "20230001")
    aluno.exibir_informacoes()
    aluno.exibir_matricula()

if __name__ == "__main__":
    main()
```

Explicação: Classe Pessoa: Define os atributos nome e cpf, além de um método exibir_informacoes() para exibir essas informações.
Classe Aluno: Herda de Pessoa, adiciona o atributo matricula e o método exibir_matricula() para exibir o número de matrícula.
Função main(): Cria uma instância de Aluno, define os valores para os atributos e exibe as informações usando os métodos herdados e específicos.

Sobrescrita de Métodos

Exercício 169: Sobrescrita de Método na Classe Animal

Enunciado: Crie uma classe Animal com o método som() que exibe "O animal está fazendo um som". Crie uma subclasse Gato que sobrescreve o método som() para exibir "O gato está miando". No método main(), crie objetos de ambas as classes e chame o método som().

Código de Solução:

```
class Animal:
    def som(self):
        print("O animal está fazendo um som.")

class Gato(Animal):
    def som(self):
        print("O gato está miando.")

def main():
    animal = Animal()
    gato = Gato()
```

```
    animal.som()  # Exibe som genérico de animal
    gato.som()    # Exibe som específico do gato

if __name__ == "__main__":
    main()
```

Explicação: Classe Animal: Define o método som() que exibe uma mensagem genérica sobre o som de um animal. Classe Gato: Herda de Animal e sobrescreve o método som() para fornecer uma mensagem específica. Função main(): Cria instâncias de Animal e Gato e chama o método som() de ambos. Sobrescrita de método: O método som() na classe Gato substitui a versão herdada de Animal, demonstrando polimorfismo.

Exercício 170: Sobrescrita de Método na Classe Veiculo

Enunciado: Implemente uma classe Veiculo com o método acelerar() que exibe "O veículo está acelerando". Crie uma subclasse Moto que sobrescreva esse método para exibir "A moto está acelerando rapidamente". No método main(), crie um objeto de cada classe e chame o método acelerar().

Código de Solução:

```
class Veiculo:
    def acelerar(self):
        print("O veículo está acelerando.")

class Moto(Veiculo):
    def acelerar(self):
        print("A moto está acelerando rapidamente.")

def main():
    veiculo = Veiculo()
    moto = Moto()

    veiculo.acelerar()  # Exibe aceleração genérica do veículo
    moto.acelerar()     # Exibe aceleração específica da moto

if __name__ == "__main__":
    main()
```

Explicação: Classe Veiculo: Define o método acelerar() com um comportamento genérico para veículos. Classe Moto: Herda de Veiculo e sobrescreve o método acelerar() para fornecer um comportamento específico. Função main(): Cria instâncias de Veiculo e Moto e

chama o método acelerar() de cada uma, demonstrando o uso do polimorfismo. Sobrescrita de Método: O método acelerar() na classe Moto substitui a implementação herdada de Veiculo, mostrando a personalização de comportamento em subclasses.

Exercício 171: Sobrescrita de Método na Classe Pessoa

Enunciado: Crie uma classe Pessoa com o método trabalhar() que exibe "A pessoa está trabalhando". Crie uma subclasse Professor que sobrescreva o método para exibir "O professor está dando aula". No método main(), crie objetos de ambas as classes e chame o método trabalhar().

Código de Solução:

```
class Pessoa:
    def trabalhar(self):
        print("A pessoa está trabalhando.")

class Professor(Pessoa):
    def trabalhar(self):
        print("O professor está dando aula.")

def main():
    pessoa = Pessoa()
    professor = Professor()

    pessoa.trabalhar()   # Exibe trabalho genérico
    professor.trabalhar()  # Exibe trabalho específico do professor

if __name__ == "__main__":
    main()
```

Explicação: Classe Pessoa: Define o método trabalhar() com um comportamento genérico que exibe "A pessoa está trabalhando."
Classe Professor: Herda de Pessoa e sobrescreve o método trabalhar() para exibir "O professor está dando aula."
Função main(): Cria instâncias de Pessoa e Professor e demonstra o polimorfismo, onde cada classe executa sua própria versão do método trabalhar().

Exercício 172: Sobrescrita de Método na Classe Conta

Enunciado: Crie uma classe Conta com o método depositar() que exibe "Depositando na conta padrão". Crie uma subclasse ContaPoupanca que sobrescreva o método para exibir "Depositando na conta poupança". No método main(), crie objetos de ambas as classes e chame o método depositar().

Código de Solução:

```
class Conta:
    def depositar(self):
        print("Depositando na conta padrão.")

class ContaPoupanca(Conta):
    def depositar(self):
        print("Depositando na conta poupança.")

def main():
    conta = Conta()
    conta_poupanca = ContaPoupanca()

    conta.depositar()         # Exibe depósito padrão
    conta_poupanca.depositar()  # Exibe depósito específico na poupança

if __name__ == "__main__":
    main()
```

Explicação: Classe Conta: Define o método depositar() que exibe a mensagem genérica "Depositando na conta padrão."
Classe ContaPoupanca: Herda de Conta e sobrescreve o método depositar() para exibir a mensagem específica "Depositando na conta poupança."
Função main(): Demonstra o polimorfismo ao criar objetos de ambas as classes (Conta e ContaPoupanca) e chama o método depositar() em cada um.

Exercício 173: Sobrescrita de Método na Classe Produto

Enunciado: Implemente uma classe Produto com o método calcularPreco() que exibe "Calculando preço base". Crie uma subclasse Eletronico que sobrescreva o método para exibir "Calculando preço com taxa eletrônica". No método main(), crie objetos de ambas as classes e chame o método calcularPreco().

Código de Solução:

```
class Produto:
```

```
    def calcular_preco(self):
        print("Calculando preço base.")

class Eletronico(Produto):
    def calcular_preco(self):
        print("Calculando preço com taxa eletrônica.")

def main():
    produto = Produto()
    eletronico = Eletronico()

    produto.calcular_preco()      # Exibe cálculo de preço base
    eletronico.calcular_preco()   # Exibe cálculo de preço específico
para eletrônicos

if __name__ == "__main__":
    main()
```

Explicação: Classe Produto: Contém o método calcular_preco(), que exibe uma mensagem genérica sobre o cálculo de preços.
Classe Eletronico: Subclasse de Produto, que sobrescreve o método calcular_preco() para exibir uma mensagem específica sobre o cálculo de preços para produtos eletrônicos.
Função main(): Demonstra o uso de polimorfismo. Criamos instâncias de Produto e Eletronico e chamamos o método calcular_preco() para observar os comportamentos diferentes.

Polimorfismo

Exercício 174: Classe Animal com Polimorfismo

Enunciado: Crie uma classe Animal com um método fazerSom(). Implemente duas subclasses: Cachorro e Gato, cada uma sobrescrevendo o método fazerSom() para exibir sons diferentes. No método main(), crie uma lista de animais (misturando Cachorro e Gato) e utilize o polimorfismo para chamar o método fazerSom() em cada objeto da lista.

Código de Solução:

```
class Animal:
    def fazer_som(self):
        print("O animal está fazendo um som.")
```

```
class Cachorro(Animal):
    def fazer_som(self):
        print("O cachorro está latindo.")

class Gato(Animal):
    def fazer_som(self):
        print("O gato está miando.")

def main():
    animais = [Cachorro(), Gato()]  # Lista de objetos Cachorro e Gato

    for animal in animais:
        animal.fazer_som()  # Polimorfismo em ação

if __name__ == "__main__":
    main()
```

Explicação: Classe Animal: Define o método genérico fazer_som(), exibindo uma mensagem padrão. Subclasses Cachorro e Gato: Ambas sobrescrevem o método fazer_som() para exibir mensagens específicas relacionadas a seus sons.
Lista de animais: Criamos uma lista que mistura objetos de ambas as subclasses (Cachorro e Gato). Iteração e polimorfismo: Ao iterar pela lista e chamar o método fazer_som(), o Python identifica o tipo específico de cada objeto e executa o método correspondente, mostrando o comportamento correto para cada tipo de animal.

Exercício 175: Classe Funcionario e Subclasses

Enunciado: Implemente uma classe Funcionario com um método calcularSalario(). Crie duas subclasses, FuncionarioCLT e FuncionarioPJ, que sobrescrevem o método para calcular o salário de forma diferente. No método main(), crie uma lista de funcionários (misturando ambos os tipos) e utilize o polimorfismo para calcular e exibir o salário de cada funcionário.

Código de Solução:

```
class Funcionario:
    def calcular_salario(self):
        print("Calculando salário base.")

class FuncionarioCLT(Funcionario):
    def calcular_salario(self):
        print("Salário CLT calculado com base no salário fixo e
```

```
benefícios.")

class FuncionarioPJ(Funcionario):
    def calcular_salario(self):
        print("Salário PJ calculado com base nas horas trabalhadas.")

def main():
    funcionarios = [FuncionarioCLT(), FuncionarioPJ()]  # Lista com
ambos os tipos de funcionários

    for funcionario in funcionarios:
        funcionario.calcular_salario()  # Polimorfismo em ação

if __name__ == "__main__":
    main()
```

Explicação: Classe Funcionario:
Define o método genérico calcular_salario() com uma mensagem padrão.
Subclasses FuncionarioCLT e FuncionarioPJ:
Sobrescrevem o método calcular_salario() para fornecer cálculos específicos a cada tipo de contratação.
Lista de funcionários:
Contém objetos das duas subclasses (FuncionarioCLT e FuncionarioPJ).
Iteração e polimorfismo:
O loop percorre a lista e chama o método calcular_salario() para cada funcionário. O Python utiliza o método correto baseado no tipo do objeto.

Exercício 176: Classe Veiculo e Subclasses com Polimorfismo

Enunciado: Crie uma classe Veiculo com um método abastecer(). Implemente as subclasses Carro e Moto, onde cada uma sobrescreve abastecer() de forma diferente (por exemplo, litros diferentes para encher o tanque). No método main(), utilize polimorfismo para criar uma lista de veículos e chame abastecer() para cada veículo.

Código de Solução:

```
class Veiculo:
    def abastecer(self):
        print("Abastecendo o veículo.")

class Carro(Veiculo):
    def abastecer(self):
        print("Abastecendo o carro com 50 litros de combustível.")
```

```
class Moto(Veiculo):
    def abastecer(self):
        print("Abastecendo a moto com 15 litros de combustível.")

def main():
    veiculos = [Carro(), Moto()]  # Lista contendo objetos de Carro e
Moto

    for veiculo in veiculos:
        veiculo.abastecer()  # Polimorfismo em ação

if __name__ == "__main__":
    main()
```

Explicação: Classe Veiculo: Define o método genérico abastecer() com uma mensagem padrão.
Subclasses Carro e Moto: Sobrescrevem o método abastecer() para fornecer uma quantidade específica de combustível ao abastecer.
Lista de veículos: Armazena objetos das subclasses (Carro e Moto).
Iteração e polimorfismo: O loop percorre a lista e chama o método abastecer() para cada veículo, com o Python selecionando o método correto com base no tipo do objeto.

Exercício 177: Classe Forma e Subclasses com Cálculo de Área

Enunciado: Implemente uma classe abstrata Forma com um método abstrato calcularArea(). Crie as subclasses Circulo e Retangulo, cada uma com a implementação do método calcularArea(). No método main(), crie uma lista de formas e, usando polimorfismo, calcule e exiba a área de cada uma.

Código de Solução:

```
from abc import ABC, abstractmethod
import math

class Forma(ABC):
    @abstractmethod
    def calcular_area(self):
        pass

class Circulo(Forma):
    def __init__(self, raio):
        self.raio = raio
```

```
    def calcular_area(self):
        return math.pi * self.raio ** 2

class Retangulo(Forma):
    def __init__(self, largura, altura):
        self.largura = largura
        self.altura = altura

    def calcular_area(self):
        return self.largura * self.altura

def main():
    formas = [Circulo(5), Retangulo(4, 6)]  # Lista contendo objetos de
Circulo e Retangulo

    for forma in formas:
        print(f"Área: {forma.calcular_area()}")

if __name__ == "__main__":
    main()
```

Explicação: Classe abstrata Forma: A classe Forma é abstrata e contém o método abstrato calcular_area(), que deve ser implementado por todas as subclasses.
Subclasses Circulo e Retangulo: A classe Circulo recebe o valor do raio e implementa calcular_area() usando a fórmula da área do círculo.
A classe Retangulo recebe a largura e altura, implementando calcular_area() com a fórmula da área do retângulo.
Método main(): Criamos uma lista contendo instâncias de Circulo e Retangulo.
Usamos polimorfismo para iterar sobre a lista e chamar o método calcular_area() de cada forma, exibindo o resultado.

Exercício 178: Interface Pagamento e Implementações

Enunciado: Crie uma interface Pagamento com um método processarPagamento(). Implemente duas classes que implementam essa interface: PagamentoCartao e PagamentoBoleto. No método main(), crie uma lista de diferentes tipos de pagamento e utilize o polimorfismo para processar cada um deles chamando o método processarPagamento().

Código de Solução:

```
from abc import ABC, abstractmethod
```

```python
class Pagamento(ABC):
    @abstractmethod
    def processar_pagamento(self):
        pass

class PagamentoCartao(Pagamento):
    def processar_pagamento(self):
        print("Processando pagamento via cartão de crédito.")

class PagamentoBoleto(Pagamento):
    def processar_pagamento(self):
        print("Processando pagamento via boleto bancário.")

def main():
    pagamentos = [PagamentoCartao(), PagamentoBoleto()]  # Lista de
diferentes tipos de pagamento

    for pagamento in pagamentos:
        pagamento.processar_pagamento()

if __name__ == "__main__":
    main()
```

Explicação: Interface Pagamento:
Pagamento em Python, que define o método abstrato processar_pagamento(). Qualquer classe que a implemente deve fornecer uma implementação desse método.
Classes PagamentoCartao e PagamentoBoleto: Ambas as classes implementam o método processar_pagamento() de maneira específica: a classe PagamentoCartao processa o pagamento via cartão de crédito, e a classe PagamentoBoleto processa o pagamento via boleto bancário.
Método main(): Criamos uma lista pagamentos contendo instâncias de PagamentoCartao e PagamentoBoleto. Usamos polimorfismo para iterar sobre a lista e chamar o método processar_pagamento() de cada tipo de pagamento.

Conclusão do Capítulo 8: Herança e Polimorfismo

Neste capítulo, vimos dois conceitos fundamentais da Programação Orientada a Objetos (POO): herança e polimorfismo.

Aprendemos como a herança permite o reaproveitamento de código, criando hierarquias entre classes, e como a sobrescrita de métodos possibilita a personalização de comportamento nas subclasses.

O polimorfismo complementa esses conceitos, permitindo o tratamento uniforme de objetos de diferentes tipos, simplificando a estrutura do código e promovendo sua flexibilidade.

Introdução ao Capítulo 9: Manipulação de Strings

No Capítulo 9, exploraremos a manipulação de strings, uma das tarefas mais comuns na programação.

As strings representam sequências de caracteres e são amplamente utilizadas em diversas aplicações, desde processamento de textos até interação com o usuário. Começaremos com a manipulação básica, como concatenar, comparar e extrair partes de strings.

Em seguida, abordaremos os métodos da classe str, que oferecem uma variedade de ferramentas poderosas para trabalhar com textos de forma eficiente. Ao dominar esses conceitos, você será capaz de lidar com manipulações textuais de forma precisa e eficaz.

Manipulação básica de Strings

Exercício 179: Exibir String

Enunciado: Escreva um programa que receba uma string do usuário e exiba a mesma string no console.

Código de Solução:

```
# Exibir String

# Recebe uma string do usuário e exibe no console
def exibir_string():
    texto = input("Digite uma string: ")
    print("Você digitou:", texto)

# Função principal para executar o programa
if __name__ == "__main__":
    exibir_string()
```

Explicação: Utilizamos a função input() para capturar a entrada do usuário, que retorna a string digitada. Em seguida, a função print() é usada para exibir a string no console.

Exercício 180: Comprimento da String

Enunciado: Crie um programa que receba uma string e exiba o número de caracteres que ela contém.

Código de Solução:

```
# Comprimento da String

# Recebe uma string do usuário e exibe o número de caracteres
def comprimento_string():
    texto = input("Digite uma string: ")
    print("Comprimento da string:", len(texto))

# Função principal para executar o programa
if __name__ == "__main__":
    comprimento_string()
```

Explicação: Utilizamos a função input() para capturar a entrada do usuário. A função len() é utilizada para contar o número de caracteres na string, incluindo espaços. O resultado é exibido com a função print().

Exercício 181: Concatenar Strings

Enunciado: Implemente um programa que receba duas strings do usuário e exiba a concatenação delas.

Código de Solução:

```
# Concatenar Strings

# Recebe duas strings do usuário e exibe a concatenação delas
def concatenar_strings():
    string1 = input("Digite a primeira string: ")
    string2 = input("Digite a segunda string: ")

    resultado = string1 + string2
    print("Resultado da concatenação:", resultado)

# Função principal para executar o programa
if __name__ == "__main__":
```

```
    concatenar_strings()
```

Explicação: Utilizamos a função input() para receber as duas strings do usuário. O operador + é utilizado para concatenar as duas strings. A concatenação resultante é exibida com a função print().

Exercício 182: Comparar Strings

Enunciado: Escreva um programa que receba duas strings e compare se elas são iguais, exibindo uma mensagem no console.

Código de Solução:

```
# Comparar Strings

# Recebe duas strings do usuário e verifica se são iguais
def comparar_strings():
    string1 = input("Digite a primeira string: ")
    string2 = input("Digite a segunda string: ")

    if string1 == string2:
        print("As strings são iguais.")
    else:
        print("As strings são diferentes.")

# Função principal para executar o programa
if __name__ == "__main__":
    comparar_strings()
```

Explicação: Utilizamos a função input() para receber as duas strings do usuário. O operador == é usado para comparar o conteúdo das strings. Se as strings forem iguais, é exibida a mensagem "As strings são iguais", caso contrário, "As strings são diferentes".

Exercício 183: Converter para Maiúsculas

Enunciado: Crie um programa que receba uma string e exiba a versão da string convertida para letras maiúsculas.

Código de Solução:

```python
# Converter para Maiúsculas

# Recebe uma string do usuário e converte para letras maiúsculas
def converter_maiusculas():
    texto = input("Digite uma string: ")
    texto_maiusculas = texto.upper()
    print("String em maiúsculas:", texto_maiusculas)

# Função principal para executar o programa
if __name__ == "__main__":
    converter_maiusculas()
```

Explicação: O método upper() em Python é usado para converter todos os caracteres da string para letras maiúsculas. Utilizamos a função input() para capturar a entrada do usuário. O programa recebe a string, converte para maiúsculas usando upper() e exibe o resultado no console.

Exercício 184: Substring

Enunciado: Escreva um programa que receba uma string e um número inteiro do usuário. O programa deve exibir os primeiros N caracteres da string, onde N é o número fornecido.

Código de Solução:

```python
# Exibir os primeiros N caracteres de uma string

def exibir_substring():
    texto = input("Digite uma string: ")
    n = int(input("Digite um número inteiro: "))

    substring = texto[:n]
    print(f"Primeiros {n} caracteres: {substring}")

# Função principal para executar o programa
if __name__ == "__main__":
    exibir_substring()
```

Explicação: Em Python, essa funcionalidade pode ser replicada com o uso de slicing (texto[:n]). O programa recebe uma string e um número inteiro do usuário usando input(). Utilizamos slicing para obter os primeiros n caracteres da string e exibimos o resultado. Esse método é simples e eficiente para manipular strings em Python.

Exercício 185: Remover Espaços

Enunciado: Implemente um programa que receba uma string do usuário e remova todos os espaços em branco, exibindo o resultado.

Código de Solução:

```
# Programa para remover espaços em branco de uma string

def remover_espacos():
    texto = input("Digite uma string: ")
    texto_sem_espacos = texto.replace(" ", "")
    print(f"String sem espaços: {texto_sem_espacos}")

# Função principal para executar o programa
if __name__ == "__main__":
    remover_espacos()
```

Explicação: O método replace(" ", "") em Python funciona substituindo todos os espaços por uma string vazia. A função input() é usada para receber a string do usuário. O resultado, com os espaços removidos, é exibido ao final.

Exercício 186: Contar Ocorrências de um Caractere

Enunciado: Crie um programa que receba uma string e um caractere. O programa deve contar e exibir quantas vezes o caractere aparece na string.

Código de Solução:

```
# Programa para contar as ocorrências de um caractere em uma string

def contar_ocorrencias():
    texto = input("Digite uma string: ")
    caractere = input("Digite um caractere: ")

    if len(caractere) != 1:
        print("Por favor, insira apenas um caractere.")
```

```
        return

    contador = 0
    for char in texto:
        if char == caractere:
            contador += 1

    print(f"O caractere '{caractere}' aparece {contador} vezes.")

# Função principal para executar o programa
if __name__ == "__main__":
    contar_ocorrencias()
```

Explicação: O programa usa input() para receber a string e o caractere do usuário. Uma validação simples verifica se o usuário inseriu apenas um caractere. A string é percorrida com um loop for e cada caractere é comparado com o caractere fornecido. O contador é incrementado a cada correspondência encontrada. O resultado é exibido com o número total de ocorrências do caractere.

Exercício 187: Inverter String

Enunciado: Escreva um programa que receba uma string e exiba a versão invertida da string.

Código de Solução:

```
# Programa para inverter uma string

def inverter_string():
    texto = input("Digite uma string: ")
    texto_invertido = texto[::-1]  # Inverte a string usando slicing
    print(f"String invertida: {texto_invertido}")

# Função principal para executar o programa
if __name__ == "__main__":
    inverter_string()
```

Explicação: O programa usa input() para receber a string do usuário. A string é invertida utilizando a técnica de slicing ([::-1]), que percorre a string de trás para frente. O resultado é exibido no console com a string invertida.

Exercício 188: Substituir Caracteres

Enunciado: Implemente um programa que receba uma string e substitua todas as ocorrências de um caractere por outro caractere, fornecidos pelo usuário.

Código de Solução:

```
# Programa para substituir caracteres em uma string

def substituir_caracteres():
    texto = input("Digite uma string: ")
    antigo = input("Digite o caractere a ser substituído: ")
    novo = input("Digite o novo caractere: ")

    # Substitui todas as ocorrências do caractere antigo pelo novo
    resultado = texto.replace(antigo, novo)

    print(f"Resultado após substituição: {resultado}")

# Função principal para executar o programa
if __name__ == "__main__":
    substituir_caracteres()
```

Explicação: O programa usa input() para capturar: A string original. O caractere a ser substituído. O novo caractere que ocupará o lugar do anterior. O método replace() é usado para substituir todas as ocorrências do caractere antigo pelo novo na string. O resultado final é exibido no console.

Métodos da Classe String

Exercício 189: Usando o Método charAt()

Enunciado: Escreva um programa que receba uma string e um número inteiro do usuário. O programa deve exibir o caractere da string que está na posição indicada pelo número.

Código de Solução:

```
# Programa para exibir o caractere na posição indicada pelo número
```

```
def caractere_na_posicao():
    texto = input("Digite uma string: ")
    indice = int(input("Digite um número inteiro: "))

    # Verifica se o índice está dentro do intervalo válido
    if 0 <= indice < len(texto):
        print(f"Caractere na posição {indice}: {texto[indice]}")
    else:
        print("Índice fora do intervalo válido.")

# Função principal para executar o programa
if __name__ == "__main__":
    caractere_na_posicao()
```

Explicação:O programa usa input() para capturar: A string fornecida pelo usuário.
O índice (número inteiro) que indica a posição do caractere. A função len() é usada para verificar o tamanho da string e garantir que o índice esteja dentro do intervalo válido (0 até len(texto) - 1). O caractere na posição especificada é exibido se o índice for válido, caso contrário, o programa informa que o índice está fora do intervalo.

Exercício 190: Usando o Método equalsIgnoreCase()

Enunciado: Implemente um programa que receba duas strings do usuário e verifique se elas são iguais, ignorando a diferença entre maiúsculas e minúsculas.

Código de Solução:

```
# Programa para verificar se duas strings são iguais, ignorando a
diferença entre maiúsculas e minúsculas

def comparar_strings():
    string1 = input("Digite a primeira string: ")
    string2 = input("Digite a segunda string: ")

    # Compara as strings ignorando a diferença entre maiúsculas e
minúsculas
    if string1.lower() == string2.lower():
        print("As strings são iguais (ignorando
maiúsculas/minúsculas).")
    else:
        print("As strings são diferentes.")
```

```
# Função principal para executar o programa
if __name__ == "__main__":
    comparar_strings()
```

Explicação: O programa usa input() para capturar as duas strings fornecidas pelo usuário. A comparação entre as strings é feita utilizando o método lower(), que converte ambas as strings para minúsculas, ignorando as diferenças entre maiúsculas e minúsculas.
Se as strings forem iguais (após a conversão), a mensagem "As strings são iguais" é exibida; caso contrário, a mensagem "As strings são diferentes" é mostrada.

Exercício 191: Usando o Método contains()

Enunciado: Escreva um programa que receba uma string e uma palavra. O programa deve verificar se a palavra está contida na string e exibir uma mensagem correspondente.

Código de Solução:

```
# Programa para verificar se uma palavra está contida em uma string

def verificar_palavra():
    texto = input("Digite uma string: ")
    palavra = input("Digite uma palavra: ")

    if palavra in texto:
        print(f'A palavra "{palavra}" está contida na string.')
    else:
        print(f'A palavra "{palavra}" não está contida na string.')

# Função principal para executar o programa
if __name__ == "__main__":
    verificar_palavra()
```

Explicação: O programa usa input() para capturar a string e a palavra fornecidas pelo usuário. A verificação de se a palavra está na string é feita com a expressão palavra in texto, que retorna True se a palavra estiver contida na string. Dependendo do resultado da verificação, uma mensagem correspondente é exibida.

Exercício 192: Usando o Método indexOf()

Enunciado: Implemente um programa que receba uma string e um caractere. O programa deve exibir a posição da primeira ocorrência do caractere na string, ou uma mensagem informando que o caractere não foi encontrado.

Código de Solução:

```
# Programa para verificar a posição da primeira ocorrência de um
caractere em uma string

def encontrar_posicao():
    texto = input("Digite uma string: ")
    caractere = input("Digite um caractere: ")

    posicao = texto.find(caractere)

    if posicao != -1:
        print(f"O caractere '{caractere}' aparece pela primeira vez na
posição: {posicao}")
    else:
        print(f"O caractere '{caractere}' não foi encontrado na
string.")

# Função principal para executar o programa
if __name__ == "__main__":
    encontrar_posicao()
```

Explicação: O programa usa input() para capturar a string e o caractere fornecido pelo usuário. O método find() é utilizado para encontrar a posição da primeira ocorrência do caractere na string. Se o caractere não for encontrado, find() retorna -1. Uma mensagem é exibida com a posição do caractere ou informando que ele não foi encontrado, conforme o resultado da verificação.

Exercício 193: Usando o Método startsWith()

Enunciado: Crie um programa que receba uma string e verifique se ela começa com uma determinada palavra, fornecida pelo usuário. Exiba uma mensagem indicando o resultado.

Código de Solução:

```
# Programa para verificar se uma string começa com uma palavra fornecida
pelo usuário
```

```
def verificar_inicio():
    texto = input("Digite uma string: ")
    palavra = input("Digite a palavra inicial: ")

    if texto.startswith(palavra):
        print(f"A string começa com \"{palavra}\".")
    else:
        print(f"A string não começa com \"{palavra}\".")

# Função principal para executar o programa
if __name__ == "__main__":
    verificar_inicio()
```

Explicação: O programa usa input() para capturar a string e a palavra fornecida pelo usuário. O método startswith() é utilizado para verificar se a string começa com a palavra fornecida. O programa exibe uma mensagem indicando se a string começa ou não com a palavra fornecida pelo usuário.

Exercício 194: Usando o Método endsWith()

Enunciado: Desenvolva um programa que receba uma string e verifique se ela termina com uma determinada palavra, fornecida pelo usuário. Exiba o resultado no console.

Código de Solução:

```
# Programa para verificar se uma string termina com uma palavra
fornecida pelo usuário

def verificar_fim():
    texto = input("Digite uma string: ")
    palavra = input("Digite a palavra final: ")

    if texto.endswith(palavra):
        print(f"A string termina com \"{palavra}\".")
    else:
        print(f"A string não termina com \"{palavra}\".")

# Função principal para executar o programa
if __name__ == "__main__":
    verificar_fim()
```

Explicação: O programa usa input() para capturar a string e a palavra fornecida pelo usuário. O método endswith() é utilizado para verificar se a string termina com a palavra

fornecida. O programa exibe uma mensagem indicando se a string termina ou não com a palavra fornecida pelo usuário.

Exercício 195: Usando o Método replaceAll()

Enunciado: Implemente um programa que receba uma string e substitua todas as ocorrências de uma palavra por outra, ambas fornecidas pelo usuário.

Código de Solução:

```
# Programa para substituir todas as ocorrências de uma palavra por outra em uma string

def substituir_palavra():
    texto = input("Digite uma string: ")
    antiga = input("Digite a palavra a ser substituída: ")
    nova = input("Digite a nova palavra: ")

    resultado = texto.replace(antiga, nova)
    print("Resultado:", resultado)

# Função principal para executar o programa
if __name__ == "__main__":
    substituir_palavra()
```

Explicação: O programa utiliza input() para capturar a string e as palavras fornecidas pelo usuário. O método replace() é usado para substituir todas as ocorrências da palavra antiga pela nova. O resultado da substituição é exibido no console.

Exercício 196: Usando o Método split()

Enunciado: Escreva um programa que receba uma string e separe as palavras usando espaços como delimitadores. Exiba cada palavra em uma linha separada.

Código de Solução:

```
# Programa para separar palavras de uma string e exibi-las uma por linha

def separar_palavras():
```

```
    texto = input("Digite uma string: ")

    palavras = texto.split(" ")

    print("Palavras separadas:")
    for palavra in palavras:
        print(palavra)

# Função principal para executar o programa
if __name__ == "__main__":
    separar_palavras()
```

Explicação: O programa captura a string com input(). O método split(" ") é usado para dividir a string em uma lista de palavras, com base no espaço como delimitador. Cada palavra é impressa em uma linha separada no console.

Exercício 197: Usando o Método trim()

Enunciado: Implemente um programa que receba uma string com espaços em excesso no início e no fim, e exiba a string "limpa", sem esses espaços, utilizando o método trim().

Código de Solução:

```
# Programa para remover espaços em excesso no início e no final de uma
string

def limpar_espacos():
    texto = input("Digite uma string com espaços em excesso: ")

    texto_limpo = texto.strip()
    print(f"String sem espaços em excesso: \"{texto_limpo}\"")

# Função principal para executar o programa
if __name__ == "__main__":
    limpar_espacos()
```

Explicação: A função input() captura a string do usuário. O método strip() é usado para remover espaços em branco no início e no final da string. O programa exibe a string "limpa" com uma mensagem formatada usando f-string para incluir aspas ao redor da saída.

Exercício 198: Usando o Método compareTo()

Enunciado: Escreva um programa que compare duas strings fornecidas pelo usuário e exiba o resultado da comparação lexicográfica entre elas.

Código de Solução:

```
# Programa para comparar duas strings lexicograficamente

def comparar_strings():
    string1 = input("Digite a primeira string: ")
    string2 = input("Digite a segunda string: ")

    if string1 < string2:
        print(f"\"{string1}\" vem antes de \"{string2}\"
lexicograficamente.")
    elif string1 > string2:
        print(f"\"{string1}\" vem depois de \"{string2}\"
lexicograficamente.")
    else:
        print("As strings são iguais lexicograficamente.")

# Função principal para executar o programa
if __name__ == "__main__":
    comparar_strings()
```

Explicação: O operador < é usado para verificar se string1 vem antes de string2 lexicograficamente. O operador > é usado para verificar se string1 vem depois de string2.
O operador == é usado para verificar se as strings são iguais.
O programa exibe a mensagem apropriada com base na comparação.

Conclusão do Capítulo 9: Manipulação de Strings

Neste capítulo, exploramos a manipulação de strings, abordando tanto operações básicas quanto o uso dos métodos da classe str.

Na parte de manipulação básica, aprendemos a lidar com tarefas essenciais, como concatenar, comparar e modificar strings. Já nos métodos da classe str, vimos como utilizar funcionalidades poderosas para operações mais avançadas, como busca, substituição e divisão de strings.

Ao dominar essas técnicas, você está apto a manipular e processar textos de maneira eficiente, essencial para uma grande variedade de aplicações na programação.

Introdução ao Capítulo 10: Desafios

Neste capítulo, você enfrentará desafios práticos que irão testar os conhecimentos adquiridos ao longo do curso.

Os exercícios serão mais complexos e exigirão uma combinação de habilidades, incluindo lógica de programação, manipulação de dados e conceitos de orientação a objetos.

Estes desafios têm o objetivo de consolidar o que foi aprendido, incentivando o pensamento crítico e a resolução de problemas de forma criativa e eficiente.

Prepare-se para aplicar suas habilidades de forma prática e desafiadora, expandindo ainda mais seu domínio da programação.

Desafio 1: Calculadora Completa

Enunciado: Crie uma calculadora que permita ao usuário realizar operações matemáticas básicas (soma, subtração, multiplicação e divisão). O usuário deve poder inserir dois números e escolher a operação desejada.

Solução:

```
# Função para realizar a operação de acordo com a escolha do usuário
def calculadora():
    try:
        num1 = float(input("Digite o primeiro número: "))
        num2 = float(input("Digite o segundo número: "))

        print("Escolha a operação:")
        print("1: Soma")
        print("2: Subtração")
        print("3: Multiplicação")
        print("4: Divisão")

        opcao = int(input("Escolha uma opção (1/2/3/4): "))

        if opcao == 1:
            resultado = num1 + num2
            print(f"Resultado: {resultado}")
```

```
        elif opcao == 2:
            resultado = num1 - num2
            print(f"Resultado: {resultado}")
        elif opcao == 3:
            resultado = num1 * num2
            print(f"Resultado: {resultado}")
        elif opcao == 4:
            if num2 != 0:
                resultado = num1 / num2
                print(f"Resultado: {resultado}")
            else:
                print("Erro: Divisão por zero.")
        else:
            print("Operação inválida.")

    except ValueError:
        print("Erro: Entrada inválida. Por favor, insira números
válidos.")

# Chama a função principal para executar o programa
if __name__ == "__main__":
    calculadora()
```

Explicação:O programa começa solicitando dois números ao usuário. Em seguida, apresenta as opções de operações (soma, subtração, multiplicação e divisão). O usuário escolhe uma operação e o programa realiza a operação correspondente. Para a divisão, há uma verificação para evitar a divisão por zero. Se o usuário inserir uma entrada inválida (como texto ao invés de números), o programa trata o erro com um bloco try-except para evitar que o programa quebre.

Desafio 2: Verificador de Palíndromo

Enunciado: Implemente um programa que receba uma palavra ou frase e verifique se ela é um palíndromo (se lê da mesma forma de frente para trás).

Solução:

```
def verificar_palindromo():
    texto = input("Digite uma palavra ou frase: ").replace(" ",
"").lower()
    invertido = texto[::-1]  # Invertendo a string
```

```
    if texto == invertido:
        print("É um palíndromo.")
    else:
        print("Não é um palíndromo.")

if __name__ == "__main__":
    verificar_palindromo()
```

Explicação: Recebendo o texto: Usamos input() para obter uma palavra ou frase do usuário. Removendo espaços e ajustando para minúsculas: O método replace(" ", "") remove os espaços em branco, e lower() converte todos os caracteres para letras minúsculas. Invertendo a string: O fatiamento [::-1] cria uma versão invertida do texto. Comparação: Verificamos se a string original (modificada) é igual à sua versão invertida. Se forem iguais, a entrada é um palíndromo.
Estrutura principal: A função é chamada dentro de um bloco if __name__ == "__main__" para garantir que ela só seja executada se o script for executado diretamente.

Desafio 3: Contador de Palavras

Enunciado: Crie um programa que receba uma string e conte quantas palavras existem nela.

Solução:

```
def contar_palavras():
    frase = input("Digite uma frase: ").strip()
    palavras = frase.split()  # Divide a frase em palavras com base nos espaços
    print(f"A frase contém {len(palavras)} palavras.")

if __name__ == "__main__":
    contar_palavras()
```

Explicação: Recebendo a string: Utilizamos input() para capturar a frase digitada pelo usuário.
Removendo espaços extras: O método strip() remove espaços em branco no início e no fim da string.
Dividindo em palavras: O método split() divide a string em uma lista de palavras, considerando espaços como separadores. Ele ignora automaticamente múltiplos espaços consecutivos.
Contando as palavras: A função len() é usada para contar o número de elementos na lista palavras.

Exibindo o resultado: O número total de palavras é exibido.

Desafio 4: Jogo da Adivinhação

Enunciado: Desenvolva um jogo em que o programa escolhe um número aleatório entre 1 e 100, e o usuário deve tentar adivinhar esse número. O programa deve fornecer dicas se o palpite for maior ou menor que o número secreto.

Solução:

```
import random

def jogo_adivinhacao():
    numero_secreto = random.randint(1, 100)  # Gera um número aleatório
entre 1 e 100
    acertou = False

    print("Tente adivinhar o número entre 1 e 100!")

    while not acertou:
        try:
            palpite = int(input("Digite seu palpite: "))

            if palpite == numero_secreto:
                print("Parabéns! Você acertou!")
                acertou = True
            elif palpite < numero_secreto:
                print("O número é maior.")
            else:
                print("O número é menor.")
        except ValueError:
            print("Por favor, digite um número válido.")

if __name__ == "__main__":
    jogo_adivinhacao()
```

Explicação: Gerando o número secreto:Utilizamos random.randint(1, 100) para gerar um número aleatório entre 1 e 100.
Recebendo palpites:input() é usado para capturar os palpites do jogador.
O try-except garante que entradas inválidas, como letras, não quebrem o programa.
Comparando os palpites: Se o palpite for igual ao número secreto, o jogador vence.
Se o palpite for menor, é exibida a mensagem "O número é maior." Se for maior, é exibida "O número é menor."

Repetição: O loop while continua até que o jogador acerte o número.
Mensagem de vitória: Assim que o jogador acerta, a mensagem de parabéns é exibida, e o loop é interrompido.

Desafio 5: Validador de Senha

Enunciado: Implemente um validador de senha que verifique se a senha do usuário atende aos seguintes critérios: no mínimo 8 caracteres, contém pelo menos um número, uma letra maiúscula, uma letra minúscula e um caractere especial.

Solução:

```
def validar_senha(senha):
    if len(senha) < 8:
        return False

    tem_maiuscula = any(c.isupper() for c in senha)
    tem_minuscula = any(c.islower() for c in senha)
    tem_numero = any(c.isdigit() for c in senha)
    tem_especial = any(not c.isalnum() for c in senha)

    return tem_maiuscula and tem_minuscula and tem_numero and
tem_especial


def main():
    senha = input("Digite uma senha: ")

    if validar_senha(senha):
        print("Senha válida.")
    else:
        print("Senha inválida. A senha deve ter no mínimo 8 caracteres,
conter uma letra maiúscula, uma minúscula, um número e um caractere
especial.")


if __name__ == "__main__":
    main()
```

Explicação: Função validar_senha: Verifica se a senha atende a todos os critérios:
Comprimento mínimo de 8 caracteres: Verificado com len(senha) < 8.
Presença de uma letra maiúscula: Usamos any(c.isupper() for c in senha).
Presença de uma letra minúscula: Usamos any(c.islower() for c in senha).

Presença de um número: Usamos any(c.isdigit() for c in senha).
Presença de um caractere especial: Verificado com any(not c.isalnum() for c in senha), onde isalnum() retorna False para caracteres especiais.
Função main: Captura a senha do usuário com input.
Exibe uma mensagem informando se a senha é válida ou inválida com base na validação.
Uso do if __name__ == "__main__"::
Garante que o programa principal será executado apenas se o arquivo for executado diretamente, o que é uma boa prática em Python.

Desafio 6: Ordenação de Números

Enunciado: Crie um programa que receba uma lista de números do usuário e os ordene em ordem crescente.

Solução:

```
def main():
    n = int(input("Digite a quantidade de números: "))

    numeros = []
    print("Digite os números:")
    for _ in range(n):
        numero = int(input())
        numeros.append(numero)

    numeros.sort()

    print("Números em ordem crescente:", numeros)


if __name__ == "__main__":
    main()
```

Explicação: Receber a quantidade de números: O programa solicita que o usuário informe a quantidade de números a serem inseridos com input().
Coletar os números: Um laço for é usado para coletar n números, que são adicionados a uma lista usando append().
Ordenar a lista: O método sort() é utilizado para ordenar os números em ordem crescente diretamente na lista.
Exibir o resultado: A lista ordenada é exibida no console.

Desafio 7: Contagem de Vogais

Enunciado: Escreva um programa que receba uma string e conte quantas vogais existem nela.

Solução:

```
def contar_vogais(texto):
    texto = texto.lower()  # Converte para letras minúsculas
    vogais = "aeiou"
    contagem_vogais = sum(1 for c in texto if c in vogais)
    return contagem_vogais

def main():
    texto = input("Digite uma string: ")
    total_vogais = contar_vogais(texto)
    print(f"Número de vogais: {total_vogais}")

if __name__ == "__main__":
    main()
```

Explicação: Função contar_vogais: A string é convertida para letras minúsculas com lower() para facilitar a verificação. É usada a variável vogais para armazenar todas as vogais. Um gerador é utilizado para contar quantos caracteres na string pertencem ao conjunto de vogais com sum().
Função main: Recebe a string do usuário usando input(). Chama a função contar_vogais para calcular o número de vogais. Exibe o resultado no console.
Execução do programa: A função main é chamada apenas quando o programa é executado diretamente.

Desafio 8: Fatorial Recursivo

Enunciado: Desenvolva uma função recursiva que calcule o fatorial de um número fornecido pelo usuário.

Solução:

```
def fatorial(n):
    if n == 0:
        return 1
    return n * fatorial(n - 1)

def main():
    numero = int(input("Digite um número: "))
```

```
    print(f"O fatorial de {numero} é: {fatorial(numero)}")

if __name__ == "__main__":
    main()
```

Explicação: Função fatorial: Caso base: Se n == 0, retorna 1 (o fatorial de 0 é definido como 1). Caso recursivo: Retorna n * fatorial(n - 1), chamando a função recursivamente até atingir o caso base.
Função main: Recebe um número do usuário com input(). Converte o número para inteiro com int() e chama a função fatorial. Exibe o resultado formatado no console.
Execução do programa: A função main é chamada apenas quando o programa é executado diretamente.

Desafio 9: Conversor de Temperatura

Enunciado: Implemente um conversor de temperatura que converta valores entre Celsius, Fahrenheit e Kelvin. O usuário deve escolher a unidade de origem e destino.

Solução:

```
def converter_temperatura(temp, origem, destino):
    if origem == destino:
        return temp
    # Convertendo para Celsius
    if origem == 2:  # Fahrenheit para Celsius
        temp = (temp - 32) * 5 / 9
    elif origem == 3:  # Kelvin para Celsius
        temp = temp - 273.15
    # Convertendo de Celsius para unidade destino
    if destino == 2:  # Celsius para Fahrenheit
        return (temp * 9 / 5) + 32
    elif destino == 3:  # Celsius para Kelvin
        return temp + 273.15
    return temp  # Se o destino for Celsius

def main():
    print("Digite a temperatura:")
    temperatura = float(input())

    print("Escolha a unidade de origem:")
    print("1: Celsius")
    print("2: Fahrenheit")
    print("3: Kelvin")
```

```
    origem = int(input())

    print("Escolha a unidade de destino:")
    print("1: Celsius")
    print("2: Fahrenheit")
    print("3: Kelvin")
    destino = int(input())

    resultado = converter_temperatura(temperatura, origem, destino)
    print(f"Temperatura convertida: {resultado:.2f}")

if __name__ == "__main__":
    main()
```

Explicação: Função converter_temperatura: Converte inicialmente a temperatura para Celsius se necessário. Depois, converte de Celsius para a unidade de destino.
Se a unidade de origem for a mesma da unidade de destino, retorna o valor original.
Função main: Recebe a temperatura, a unidade de origem e a unidade de destino do usuário. Chama a função converter_temperatura para realizar a conversão. Exibe o resultado com duas casas decimais usando :.2f.
Execução do programa: O programa só executa a função main quando chamado diretamente.

Desafio 10: Gerador de Senhas Aleatórias

Enunciado: Crie um programa que gere senhas aleatórias de 8 a 16 caracteres, incluindo letras, números e símbolos.

Solução:

```
import random
import string

def gerar_senha(comprimento):
    if comprimento < 8 or comprimento > 16:
        return "Comprimento inválido. Deve ser entre 8 e 16."

    caracteres = string.ascii_letters + string.digits +
"!@#$%^&*()_+-=<>?"
    senha = ''.join(random.choice(caracteres) for _ in
range(comprimento))
    return senha
```

```
def main():
    comprimento = int(input("Digite o comprimento desejado para a senha
(entre 8 e 16): "))
    resultado = gerar_senha(comprimento)
    print(f"Senha gerada: {resultado}" if comprimento >= 8 and
comprimento <= 16 else resultado)

if __name__ == "__main__":
    main()
```

Explicação: Função gerar_senha: Recebe o comprimento da senha desejada. Retorna uma mensagem de erro caso o comprimento seja inválido (fora do intervalo de 8 a 16). Gera a senha usando random.choice para selecionar caracteres aleatórios do conjunto que combina: Letras maiúsculas e minúsculas (string.ascii_letters). Dígitos (string.digits). Símbolos específicos.
Função main: Solicita ao usuário o comprimento da senha desejada. Chama a função gerar_senha e exibe o resultado.
Execução do programa: A execução é iniciada apenas quando o script é executado diretamente (if __name__ == "__main__":).

Desafio 11: Cálculo de Média de Notas

Enunciado: Desenvolva um programa que receba as notas de um aluno e calcule sua média. O programa deve determinar se o aluno está aprovado (média ≥ 7), em recuperação (média entre 5 e 6.9) ou reprovado (média < 5).

Solução:

```
def calcular_media():
    soma = 0

    quantidade_notas = int(input("Quantas notas você deseja inserir? "))

    for i in range(quantidade_notas):
        nota = float(input(f"Digite a nota {i + 1}: "))
        soma += nota

    media = soma / quantidade_notas

    print(f"Média: {media:.2f}")

    if media >= 7:
        print("Aluno aprovado.")
```

```
    elif 5 <= media < 7:
        print("Aluno em recuperação.")
    else:
        print("Aluno reprovado.")

# Executando a função
calcular_media()
```

Explicação: Entrada do número de notas: O programa solicita ao usuário quantas notas ele deseja informar.
Laço for: Um loop é usado para coletar as notas, somando cada uma delas.
Cálculo da média: A média é calculada dividindo a soma das notas pelo número total de notas.
Classificação do aluno:
Aprovado: Média maior ou igual a 7.
Recuperação: Média entre 5 e 6.9.
Reprovado: Média abaixo de 5.
Saída formatada: A média é exibida com 2 casas decimais para maior clareza.

Desafio 12: Verificador de Número Primo

Enunciado: Implemente um programa que receba um número inteiro e verifique se ele é primo.

Solução:

```
import math

def is_primo(numero):
    if numero < 2:
        return False

    for i in range(2, int(math.sqrt(numero)) + 1):
        if numero % i == 0:
            return False
    return True

def main():
    numero = int(input("Digite um número inteiro: "))

    if is_primo(numero):
        print(f"{numero} é um número primo.")
    else:
```

```
        print(f"{numero} não é um número primo.")

# Executando o programa
main()
```

Explicação: Função is_primo: Retorna False para números menores que 2, pois não são primos. Verifica a divisibilidade de numero pelos números de 2 até a raiz quadrada de numero (inclusiva), otimizando o processo.
Entrada do usuário: O programa solicita um número inteiro para verificar se é primo.
Saída: Se o número for primo, exibe a mensagem confirmando. Caso contrário, indica que não é primo.
Uso da biblioteca math: A função math.sqrt é usada para calcular a raiz quadrada, tornando o programa mais eficiente.

Desafio 13: Jogo Pedra, Papel e Tesoura

Enunciado: Crie uma versão digital do jogo Pedra, Papel e Tesoura, em que o usuário joga contra o computador. O computador deve fazer escolhas aleatórias e o programa deve determinar o vencedor de cada rodada.

Solução:

```
import random

def pedra_papel_tesoura():
    opcoes = ["Pedra", "Papel", "Tesoura"]

    print("Escolha: 0 = Pedra, 1 = Papel, 2 = Tesoura")
    escolha_usuario = int(input("Digite sua escolha: "))

    if escolha_usuario < 0 or escolha_usuario > 2:
        print("Escolha inválida! Tente novamente.")
        return

    escolha_computador = random.randint(0, 2)

    print(f"Você escolheu: {opcoes[escolha_usuario]}")
    print(f"Computador escolheu: {opcoes[escolha_computador]}")

    if escolha_usuario == escolha_computador:
        print("Empate!")
    elif (escolha_usuario == 0 and escolha_computador == 2) or \
```

```
         (escolha_usuario == 1 and escolha_computador == 0) or \
         (escolha_usuario == 2 and escolha_computador == 1):
        print("Você venceu!")
    else:
        print("Você perdeu!")

# Executando o jogo
pedra_papel_tesoura()
```

Explicação: Lista opções: Contém as opções possíveis: "Pedra", "Papel" e "Tesoura".
Entrada do usuário: O jogador escolhe sua opção digitando um número (0, 1 ou 2).
Escolha do computador: O computador escolhe aleatoriamente uma opção usando random.randint.
Regras de decisão: O programa verifica se houve empate. Se o jogador vencer, uma mensagem é exibida. Caso contrário, o jogador perde.
Validação da entrada: Caso o jogador insira um número fora do intervalo válido, uma mensagem de erro é exibida.

Desafio 14: Conversor de Moedas

Enunciado: Desenvolva um conversor de moedas que permita ao usuário converter valores entre diferentes moedas (por exemplo, de real para dólar, de euro para iene, etc.), utilizando taxas de câmbio fixas.

Solução:

```
def converter_moeda(opcao, valor):
    if opcao == 1:  # Real para Dólar
        return valor / 5.20
    elif opcao == 2:  # Real para Euro
        return valor / 5.90
    elif opcao == 3:  # Dólar para Real
        return valor * 5.20
    elif opcao == 4:  # Euro para Real
        return valor * 5.90
    else:
        print("Opção inválida")
        return 0

def conversor_moedas():
    print("Conversor de Moedas")
    print("1: Real para Dólar")
    print("2: Real para Euro")
    print("3: Dólar para Real")
```

```
    print("4: Euro para Real")
    opcao = int(input("Escolha a conversão: "))

    if opcao < 1 or opcao > 4:
        print("Opção inválida. Tente novamente.")
        return

    valor = float(input("Digite o valor: "))
    resultado = converter_moeda(opcao, valor)

    if resultado != 0:
        print(f"Valor convertido: {resultado:.2f}")

# Executando o conversor
conversor_moedas()
```

Explicação: Função converter_moeda: Realiza a conversão com base na taxa de câmbio fixa fornecida no enunciado:
Real para Dólar: divide o valor por 5.20.
Real para Euro: divide o valor por 5.90.
Dólar para Real: multiplica o valor por 5.20.
Euro para Real: multiplica o valor por 5.90.
Função conversor_moedas: Mostra as opções disponíveis ao usuário. Lê a opção escolhida e valida se está dentro do intervalo permitido. Lê o valor a ser convertido.
Chama a função converter_moeda e exibe o resultado formatado com 2 casas decimais.
Validação: O programa valida a opção escolhida e imprime "Opção inválida" se o valor estiver fora do intervalo 1-4.

Desafio 15: Contagem Regressiva

Enunciado: Implemente um programa que faça uma contagem regressiva de um número fornecido pelo usuário até 0, exibindo cada número no console.

Solução:

```
def contagem_regressiva():
    numero = int(input("Digite um número para iniciar a contagem
regressiva: "))

    for i in range(numero, -1, -1):
        print(i)

    print("Contagem regressiva finalizada!")
```

```
# Executando o programa
contagem_regressiva()
```

Explicação: Função contagem_regressiva: Solicita ao usuário que digite um número inteiro para iniciar a contagem regressiva. Usa a função range para iterar do número fornecido até 0, diminuindo de 1 em 1 (-1 é o passo do range).
Laço for: Itera pelos números de forma decrescente, incluindo o zero. Exibe cada número no console.
Mensagem de conclusão: Após a contagem, uma mensagem é exibida informando que a contagem regressiva foi finalizada.

Desafio 16: Tabuada Automática

Enunciado: Crie um programa que exiba a tabuada de um número fornecido pelo usuário, de 1 a 10.

Solução:

```
def exibir_tabuada():
    numero = int(input("Digite um número para exibir sua tabuada: "))

    print(f"Tabuada do {numero}:")
    for i in range(1, 11):
        print(f"{numero} x {i} = {numero * i}")

# Executando o programa
exibir_tabuada()
```

Explicação: Função exibir_tabuada: Solicita ao usuário um número inteiro para calcular a tabuada. Usa a função range(1, 11) para iterar pelos números de 1 a 10.
Laço for: Multiplica o número fornecido pelo usuário pelos valores de 1 a 10. Exibe cada resultado no formato número x multiplicador = resultado.
Formato dinâmico: Utiliza f-strings para formatar e exibir as multiplicações de forma clara e legível.

Desafio 17: Soma dos Dígitos de um Número

Enunciado: Desenvolva um programa que receba um número inteiro e calcule a soma dos dígitos desse número.

Solução:

```python
def soma_digitos():
    numero = int(input("Digite um número inteiro: "))
    soma = 0

    while numero != 0:
        soma += numero % 10
        numero //= 10

    print(f"Soma dos dígitos: {soma}")

# Executando o programa
soma_digitos()
```

Explicação: Entrada do Usuário: O programa solicita um número inteiro usando input() e converte para int.
Inicialização da Soma: A variável soma é iniciada como 0, onde será acumulada a soma dos dígitos.
Laço while: O laço continua enquanto o número não for 0.
Em cada iteração: numero % 10 obtém o último dígito do número. Esse dígito é somado à variável soma. numero //= 10 remove o último dígito, reduzindo o número.
Exibição do Resultado: Após o término do laço, o programa exibe a soma dos dígitos com a mensagem formatada.

Desafio 18: FizzBuzz

Enunciado: Implemente o desafio clássico FizzBuzz: exiba os números de 1 a 100, mas para múltiplos de 3 exiba "Fizz", para múltiplos de 5 exiba "Buzz", e para múltiplos de ambos exiba "FizzBuzz".

Solução:

```python
def fizz_buzz():
    for i in range(1, 101):
        if i % 3 == 0 and i % 5 == 0:
            print("FizzBuzz")
        elif i % 3 == 0:
            print("Fizz")
        elif i % 5 == 0:
            print("Buzz")
        else:
            print(i)

# Executando o programa
fizz_buzz()
```

Explicação: Laço for: O for percorre os números de 1 a 100 usando range(1, 101).
Condições:Verifica se o número atual (i) é divisível por 3 e 5 ao mesmo tempo (i % 3 == 0 and i % 5 == 0) e imprime "FizzBuzz". Caso seja divisível apenas por 3, imprime "Fizz". Caso seja divisível apenas por 5, imprime "Buzz". Caso nenhuma das condições seja verdadeira, imprime o próprio número.
Resultado no Console: O programa exibe "Fizz", "Buzz", ou "FizzBuzz" nos casos apropriados, e os números nos outros casos.

Desafio 19: Calculadora de IMC

Enunciado: Crie um programa que receba o peso e a altura do usuário e calcule seu Índice de Massa Corporal (IMC). O programa deve exibir a categoria de peso do usuário (abaixo do peso, peso normal, sobrepeso, etc.).

Solução:

```
def calcular_imc():
    # Recebe os dados do usuário
    peso = float(input("Digite seu peso (kg): "))
    altura = float(input("Digite sua altura (m): "))

    # Calcula o IMC
    imc = peso / (altura ** 2)
    print(f"Seu IMC é: {imc:.2f}")

    # Determina a categoria do IMC
    if imc < 18.5:
        print("Abaixo do peso.")
    elif 18.5 <= imc < 24.9:
        print("Peso normal.")
    elif 25 <= imc < 29.9:
        print("Sobrepeso.")
    else:
        print("Obesidade.")

# Executando o programa
calcular_imc()
```

Explicação: Entrada de dados:O programa solicita ao usuário seu peso (em kg) e altura (em metros) e os converte para o tipo float.
Cálculo do IMC: A fórmula usada é IMC = peso / altura^2. O resultado é formatado para duas casas decimais usando f"Seu IMC é: {imc:.2f}".
Classificação do IMC: O programa verifica a categoria de peso do usuário:

Abaixo do peso: IMC < 18.5
Peso normal: 18.5 ≤ IMC < 24.9
Sobrepeso: 25 ≤ IMC < 29.9
Obesidade: IMC ≥ 30
Saída: Exibe o valor do IMC e a categoria correspondente.

Desafio 20: Contador de Caracteres Únicos

Enunciado: Escreva um programa que receba uma string e conte quantos caracteres únicos (sem repetição) existem nela.

Solução:

```
def contar_caracteres_unicos():
    # Recebe a string do usuário
    texto = input("Digite uma string: ")

    # Usa um conjunto para armazenar caracteres únicos
    caracteres_unicos = set(texto)

    # Exibe o número de caracteres únicos
    print(f"Número de caracteres únicos: {len(caracteres_unicos)}")

# Executa o programa
contar_caracteres_unicos()
```

Explicação: Entrada do Usuário: O programa solicita que o usuário insira uma string.
Conjunto de Caracteres Únicos: Em Python, um conjunto (set) é utilizado para armazenar apenas valores únicos. A string é processada diretamente ao criar o conjunto, pois cada caractere da string é automaticamente inserido sem duplicatas.
Cálculo e Saída: O número de caracteres únicos é obtido usando a função len() no conjunto caracteres_unicos. O resultado é exibido ao usuário.

Conclusão do Capítulo de Desafios

Neste capítulo, você enfrentou uma série de desafios práticos que abordaram diferentes aspectos fundamentais da programação. Desde o cálculo de médias, manipulação de strings, jogos interativos, até a criação de algoritmos clássicos como o FizzBuzz, cada exercício foi projetado para reforçar sua compreensão dos conceitos essenciais.

Esses desafios não apenas exigiram uma boa lógica e raciocínio, mas também trouxeram a necessidade de aplicar estratégias eficientes, como laços de repetição, condições, manipulação de arrays, e até mesmo a implementação de funções recursivas. Além disso, você trabalhou com uma variedade de tipos de dados e operações que simulam problemas do mundo real.

Parabéns por concluir o capítulo de Desafios!

> Por fim, fica o convite para conhecer meu **Curso Completo de Python**, que vai te fazer dominar a linguagem Python!