**def comprimir\_sequencia(texto):**

**def compressao\_recursiva(indice, caracter\_atual, contador):**

**# Se chegamos ao final da string, adicionamos o resultado**

**if indice == len(texto):**

**return (str(contador) + caracter\_atual) if contador > 1 else caracter\_atual**

**# Se o caractere atual é igual ao próximo, incrementamos o contador**

**if texto[indice] == caracter\_atual:**

**return compressao\_recursiva(indice + 1, caracter\_atual, contador + 1)**

**else:**

**# Se o caractere atual não é igual ao próximo, adicionamos o resultado**

**resultado = (str(contador) + caracter\_atual) if contador > 1 else caracter\_atual**

**return resultado + compressao\_recursiva(indice + 1, texto[indice], 1)**

**if not texto: # Se a string estiver vazia, retornamos uma string vazia**

**return ""**

**# Iniciamos a recursão com o primeiro caractere e um contador de 1**

**return compressao\_recursiva(1, texto[0], 1)**

**# Exemplos de uso**

**texto\_original\_1 = "AAABBCDDDAA"**

**texto\_comprimido\_1 = comprimir\_sequencia(texto\_original\_1)**

**print(f"Original: {texto\_original\_1}")**

**print(f"Comprimido: {texto\_comprimido\_1}")**

**texto\_original\_2 = "XYZ"**

**texto\_comprimido\_2 = comprimir\_sequencia(texto\_original\_2)**

**print(f"Original: {texto\_original\_2}")**

**print(f"Comprimido: {texto\_comprimido\_2}")**

**texto\_original\_3 = "AAAAA"**

**texto\_comprimido\_3 = comprimir\_sequencia(texto\_original\_3)**

**print(f"Original: {texto\_original\_3}")**

**print(f"Comprimido: {texto\_comprimido\_3}")**

**def criptografar(texto, chave\_palavra):**

**texto\_cifrado = []**

**chave\_tamanho = len(chave\_palavra)**

**for i, char in enumerate(texto):**

**if char.isalpha(): # Verifica se o caractere é uma letra**

**# Determina o deslocamento baseado na chave**

**deslocamento = ord(chave\_palavra[i % chave\_tamanho]) % 26**

**if char.isupper(): # Se a letra é maiúscula**

**novo\_char = chr((ord(char) - ord('A') + deslocamento) % 26 + ord('A'))**

**else: # Se a letra é minúscula**

**novo\_char = chr((ord(char) - ord('a') + deslocamento) % 26 + ord('a'))**

**texto\_cifrado.append(novo\_char)**

**else:**

**texto\_cifrado.append(char) # Mantém carac não alfabéticos inalterados**

**return ''.join(texto\_cifrado)**

**def descriptografar(texto\_cifrado, chave\_palavra):**

**texto\_decifrado = []**

**chave\_tamanho = len(chave\_palavra)**

**for i, char in enumerate(texto\_cifrado):**

**if char.isalpha(): # Verifica se o caractere é uma letra**

**# Determina o deslocamento baseado na chave**

**deslocamento = ord(chave\_palavra[i % chave\_tamanho]) % 26**

**if char.isupper(): # Se a letra é maiúscula**

**novo\_char = chr((ord(char) - ord('A') - deslocamento) % 26 + ord('A'))**

**else: # Se a letra é minúscula**

**novo\_char = chr((ord(char) - ord('a') - deslocamento) % 26 + ord('a'))**

**texto\_decifrado.append(novo\_char)**

**else:**

**texto\_decifrado.append(char) # Mantém caracteres não alfabéticos inalterados**

**return ''.join(texto\_decifrado)**

**def modificar\_lista(lista\_param, valor\_param):**

**print(f" ID de lista\_param DENTRO (inicio): {id(lista\_param)}")**

**lista\_param.append(40)**

**print(f" ID de lista\_param DENTRO (apos append): {id(lista\_param)}")**

**lista\_param = [100, 200] # Reatribuição local**

**print(f" ID de lista\_param DENTRO (apos reatribuicao): {id(lista\_param)}")**

**valor\_param += 5**

**print(f" ID de valor\_param DENTRO: {id(valor\_param)}")**

**return valor\_param**

**a = 10**

**b = a**

**print(f"ID de a (inicial): {id(a)}")**

**print(f"ID de b (inicial): {id(b)}")**

**b = 20**

**print(f"ID de a (apos b=20): {id(a)}")**

**print(f"ID de b (apos b=20): {id(b)}")**

**minha\_lista = [10, 20, 30]**

**outra\_lista = minha\_lista**

**print(f"ID de minha\_lista (inicial): {id(minha\_lista)}")**

**print(f"ID de outra\_lista (inicial): {id(outra\_lista)}")**

**outra\_lista.append(35) # Modifica o objeto referenciado por ambos**

**print(f"ID de minha\_lista (apos append em outra\_lista): {id(minha\_lista)}")**

**print(f"minha\_lista (apos append em outra\_lista): {minha\_lista}")**

**c = 5**

**print(f"ID de c (antes da funcao): {id(c)}")**

**print(f"ID de minha\_lista (antes da funcao): {id(minha\_lista)}")**

**d = modificar\_lista(minha\_lista, c)**

**print(f"ID de c (depois da funcao): {id(c)}")**

**print(f"ID de d (depois da funcao): {id(d)}")**

**print(f"ID de minha\_lista (depois da funcao): {id(minha\_lista)}")**

**import os**

**def criar\_arquivo\_padrao(nome\_arquivo):**

**conteudo = """P001;Caneta Azul;10;1.50**

**P002;Caderno 10 Mat;5;15.00**

**P001;Caneta Azul;5;1.50**

**P003;Lapis Grafite;20;0.80**

**P002;Caderno 10 Mat;2;15.00**

**P001;Caneta Vermelha;8;1.60**

**"""**

**with open(nome\_arquivo, 'w') as arquivo:**

**arquivo.write(conteudo)**

**def ler\_dados\_vendas(nome\_arquivo):**

**# Verifica se o arquivo existe**

**if not os.path.exists(nome\_arquivo):**

**print(f"Arquivo {nome\_arquivo} não encontrado. Criando arquivo padrão com dados de conteudo.")**

**criar\_arquivo\_padrao(nome\_arquivo)**

**dados\_vendas = []**

**with open(nome\_arquivo, 'r') as arquivo:**

**for linha in arquivo:**

**codigo, nome, quantidade, preco = linha.strip().split(';')**

**dados\_vendas.append({**

**'codigo': codigo,**

**'nome': nome,**

**'quantidade': int(quantidade),**

**'preco': float(preco)**

**})**

**return dados\_vendas**

**def processar\_vendas(dados\_vendas):**

**dados\_consolidados = {}**

**for venda in dados\_vendas:**

**codigo = venda['codigo']**

**if codigo not in dados\_consolidados:**

**dados\_consolidados[codigo] = {**

**'nome': venda['nome'],**

**'quantidade\_total': 0,**

**'valor\_total': 0.0**

**}**

**dados\_consolidados[codigo]['quantidade\_total'] += venda['quantidade']**

**dados\_consolidados[codigo]['valor\_total'] += venda['quantidade'] \* venda['preco']**

**return dados\_consolidados**

**def gerar\_relatorio\_produto(dados\_consolidados, codigo\_produto):**

**if codigo\_produto in dados\_consolidados:**

**produto = dados\_consolidados[codigo\_produto]**

**print(f"--- Relatorio Produto: {codigo\_produto} ---")**

**print(f"Nome: {produto['nome']}")**

**print(f"Quantidade Total Vendida: {produto['quantidade\_total']}")**

**print(f"Valor Total Vendido: R$ {produto['valor\_total']:.2f}")**

**else:**

**print(f"Produto {codigo\_produto} não encontrado.")**

**def gerar\_relatorio\_geral(dados\_consolidados):**

**print("--- Relatorio Geral de Vendas ---")**

**for codigo, produto in sorted(dados\_consolidados.items()):**

**print(f"Produto: {codigo} - Nome: {produto['nome']} - Qtd: {produto['quantidade\_total']} - Total: R$ {produto['valor\_total']:.2f}")**

**def main():**

**# Define o caminho do arquivo em C:\PYTHON**

**nome\_arquivo = r'C:\PYTHON\vendas.txt'**

**dados\_vendas = ler\_dados\_vendas(nome\_arquivo)**

**dados\_consolidados = processar\_vendas(dados\_vendas)**

**print("Dados de vendas processados.")**

**while True:**

**codigo\_produto = input("Digite o codigo do produto para relatorio (ou 'TODOS', ou 'SAIR'): ").strip()**

**if codigo\_produto == 'SAIR':**

**print("Encerrando programa.")**

**break**

**elif codigo\_produto == 'TODOS':**

**gerar\_relatorio\_geral(dados\_consolidados)**

**else:**

**gerar\_relatorio\_produto(dados\_consolidados, codigo\_produto)**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

**def alterar\_dados(dict\_param: dict, num\_param: int) -> int:**

**print(f" ID de dict\_param DENTRO (inicio): {id(dict\_param)}")**

**print(f" ID de num\_param DENTRO (inicio): {id(num\_param)}")**

**# Modificando o objeto original (mutavel)**

**dict\_param['c'] = 300**

**print(f" ID de dict\_param DENTRO (apos modif.): {id(dict\_param)}")**

**# Reatribuindo a variavel local (nao afeta o objeto original)**

**num\_param = num\_param + 1**

**print(f" ID de num\_param DENTRO (apos reatrib.): {id(num\_param)}")**

**# Reatribuindo a variavel local a um novo objeto**

**dict\_param = {'x': 99, 'y': 98}**

**print(f" ID de dict\_param DENTRO (apos reatrib.): {id(dict\_param)}")**

**return num\_param**

**x = 100**

**y = x**

**print(f"ID de x (inicial): {id(x)}")**

**print(f"ID de y (inicial): {id(y)}")**

**y = 101**

**print(f"ID de x (apos y=101): {id(x)}")**

**print(f"ID de y (apos y=101): {id(y)}")**

**meu\_dict = {'a': 1, 'b': 2}**

**outro\_dict = meu\_dict**

**print(f"ID de meu\_dict (inicial): {id(meu\_dict)}")**

**print(f"ID de outro\_dict (inicial): {id(outro\_dict)}")**

**outro\_dict['b'] = 20 # Modifica o objeto referenciado por ambos**

**print(f"ID de meu\_dict (apos modif. em outro\_dict): {id(meu\_dict)}")**

**print(f"meu\_dict (apos modif. em outro\_dict): {meu\_dict}")**

**print(f"\n--- Chamando a funcao ---")**

**num\_original = 50**

**print(f"ID de meu\_dict (antes da funcao): {id(meu\_dict)}")**

**print(f"ID de num\_original (antes da funcao): {id(num\_original)}")**

**num\_retornado = alterar\_dados(meu\_dict, num\_original)**

**print(f"\n--- Apos a funcao ---")**

**print(f"ID de meu\_dict (depois da funcao): {id(meu\_dict)}")**

**print(f"ID de num\_original (depois da funcao): {id(num\_original)}")**

**print(f"ID de num\_retornado (depois da funcao): {id(num\_retornado)}")**

**print(f"\n--- Valores Finais ---")**

**print(f"Valor final de x: {x}")**

**print(f"Valor final de y: {y}")**

**print(f"Valor final de meu\_dict: {meu\_dict}")**

**print(f"Valor final de outro\_dict: {outro\_dict}")**

**print(f"Valor final de num\_original: {num\_original}")**

**print(f"Valor final de num\_retornado: {num\_retornado}")**

**# Estrutura de diretórios simulada**

**estrutura\_diretorios = {**

**"/home": {"tipo": "dir", "conteudo": ["user"]},**

**"/home/user": {"tipo": "dir", "conteudo": ["documentos", "foto.jpg", "notas.txt"]},**

**"/home/user/documentos": {"tipo": "dir", "conteudo": ["trabalho.docx", "relatorio.txt"]},**

**"/home/user/foto.jpg": {"tipo": "arq"},**

**"/home/user/notas.txt": {"tipo": "arq"},**

**"/home/user/documentos/trabalho.docx": {"tipo": "arq"},**

**"/home/user/documentos/relatorio.txt": {"tipo": "arq"}**

**}**

**# Funções auxiliares**

**def eh\_diretorio(caminho: str) -> bool:**

**return estrutura\_diretorios.get(caminho, {}).get("tipo") == "dir"**

**def eh\_arquivo(caminho: str) -> bool:**

**return estrutura\_diretorios.get(caminho, {}).get("tipo") == "arq"**

**def listar\_conteudo(caminho\_dir: str):**

**return estrutura\_diretorios.get(caminho\_dir, {}).get("conteudo", [])**

**# Função recursiva para listar arquivos com a extensão desejada**

**def listar\_arquivos\_recursivo(caminho: str, extensao: str):**

**arquivos\_encontrados = []**

**# Verifica se o caminho é um diretório**

**if eh\_diretorio(caminho):**

**# Lista o conteúdo do diretório**

**conteudo = listar\_conteudo(caminho)**

**for item in conteudo:**

**# Cria o caminho completo do item**

**caminho\_completo = f"{caminho}/{item}" if caminho != "/" else item**

**# Verifica se o item é um arquivo**

**if eh\_arquivo(caminho\_completo):**

**# Se for um arquivo e a extensão corresponder, adiciona à lista**

**if caminho\_completo.endswith(extensao):**

**arquivos\_encontrados.append(caminho\_completo)**

**# Se for um diretório, chama a função recursivamente**

**elif eh\_diretorio(caminho\_completo):**

**arquivos\_encontrados.extend(listar\_arquivos\_recursivo(caminho\_completo, extensao))**

**return arquivos\_encontrados**

**# Exemplo de uso**

**arquivos\_txt = listar\_arquivos\_recursivo("/home", ".txt")**

**print(f"Arquivos .txt encontrados: {arquivos\_txt}")**

**arquivos\_jpg = listar\_arquivos\_recursivo("/home", ".jpg")**

**print(f"Arquivos .jpg encontrados: {arquivos\_jpg}")**

**arquivos\_pdf = listar\_arquivos\_recursivo("/home", ".pdf")**

**print(f"Arquivos .pdf encontrados: {arquivos\_pdf}")**

**import os**

**def criar\_arquivo\_padrao(nome\_arquivo):**

**"""Cria um arquivo padrão com dados de exemplo."""**

**conteudo = """2025001;Ana Silva;8.5;9.0;7.5**

**2025002;Bruno Costa;5.0;6.5;7.0**

**2025003;Carla Dias;4.0;5.5;3.0**

**2025001;Ana Silva;9.5;9.0;8.0**

**2025004;Daniel Farias;10.0;9.5;9.0**

**2025002;Bruno Costa;6.0;7.0;7.0**

**"""**

**with open(nome\_arquivo, 'w') as arquivo:**

**arquivo.write(conteudo)**

**def ler\_dados\_alunos(nome\_arquivo):**

**"""Lê o arquivo e retorna um dicionário com as informações dos alunos."""**

**dados\_alunos = {}**

**if not os.path.exists(nome\_arquivo):**

**print(f"Arquivo {nome\_arquivo} não encontrado. Criando arquivo padrão com dados de exemplo.")**

**criar\_arquivo\_padrao(nome\_arquivo)**

**with open(nome\_arquivo, 'r') as arquivo:**

**for linha in arquivo:**

**matricula, nome\_aluno, nota1, nota2, nota3 = linha.strip().split(';')**

**if matricula not in dados\_alunos:**

**dados\_alunos[matricula] = {**

**'nome': nome\_aluno,**

**'notas': [float(nota1), float(nota2), float(nota3)]**

**}**

**return dados\_alunos**

**def processar\_situacao\_alunos(dados\_alunos):**

**"""Processa os dados dos alunos e retorna um dicionário com média e situação."""**

**dados\_consolidados = {}**

**for matricula, info in dados\_alunos.items():**

**media = sum(info['notas']) / len(info['notas'])**

**situacao = "Aprovado" if media >= 7.0 else "Reprovado"**

**dados\_consolidados[matricula] = {**

**'nome': info['nome'],**

**'notas': info['notas'],**

**'media': media,**

**'situacao': situacao**

**}**

**return dados\_consolidados**

**def gerar\_relatorio\_aluno(dados\_consolidados, matricula):**

**"""Imprime um relatório detalhado para um aluno específico."""**

**aluno = dados\_consolidados.get(matricula)**

**if aluno:**

**print(f"--- Relatorio do Aluno: {matricula} ---")**

**print(f"Nome: {aluno['nome']}")**

**print(f"Notas: {aluno['notas']}")**

**print(f"Média: {aluno['media']:.2f}")**

**print(f"Situação: {aluno['situacao']}")**

**else:**

**print(f"Matricula {matricula} nao encontrada.")**

**def gerar\_relatorio\_geral(dados\_consolidados, filtro):**

**"""Imprime um relatório de todos os alunos que correspondem a um filtro."""**

**filtro = filtro.upper()**

**print(f"--- Relatorio Geral: {filtro} ---")**

**valid\_situacoes = {"APROVADOS": "Aprovado", "REPROVADOS": "Reprovado", "TODOS": "TODOS"}**

**if filtro not in valid\_situacoes:**

**print("Filtro inválido.")**

**return**

**situacao\_filtrar = valid\_situacoes[filtro]**

**alunos\_filtrados = [**

**(mat, aluno) for mat, aluno in dados\_consolidados.items()**

**if situacao\_filtrar == "TODOS" or aluno['situacao'] == situacao\_filtrar**

**]**

**if not alunos\_filtrados:**

**print(f"Nenhum aluno encontrado para o filtro '{filtro}'.")**

**return**

**for matricula, aluno in sorted(alunos\_filtrados):**

**print(f"Mat: {matricula} | Nome: {aluno['nome']} | Média: {aluno['media']:.2f} | Situação: {aluno['situacao']}")**

**def main():**

**nome\_arquivo = r'C:\Python\notas.txt' # Caminho fixo para o arquivo notas.txt**

**dados\_alunos = ler\_dados\_alunos(nome\_arquivo)**

**dados\_consolidados = processar\_situacao\_alunos(dados\_alunos)**

**print("Dados dos alunos processados.")**

**while True:**

**entrada = input("Digite a matricula ou um filtro ('APROVADOS', 'REPROVADOS', 'TODOS', 'SAIR'): ").strip().upper()**

**if entrada == 'SAIR':**

**print("Encerrando programa.")**

**break**

**elif entrada in ['APROVADOS', 'REPROVADOS', 'TODOS']:**

**gerar\_relatorio\_geral(dados\_consolidados, entrada)**

**else:**

**gerar\_relatorio\_aluno(dados\_consolidados, entrada)**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

**import random**

**def escolher\_palavra(lista\_palavras):**

**"""Escolhe uma palavra aleatória da lista fornecida."""**

**return random.choice(lista\_palavras)**

**def mostrar\_palavra\_oculta(palavra\_secreta, letras\_corretas):**

**"""Retorna a representação atual da palavra, mostrando as letras corretas e ocultando as demais."""**

**return ' '.join([letra if letra in letras\_corretas else '\_' for letra in palavra\_secreta])**

**def jogar():**

**"""Função principal que orquestra o jogo da forca."""**

**lista\_palavras = ['abacaxi', 'python', 'programacao', 'desenvolvimento', 'jogo']**

**numero\_maximo\_erros = 6**

**palavra\_secreta = escolher\_palavra(lista\_palavras).upper()**

**letras\_corretas = []**

**letras\_tentadas = []**

**erros = 0**

**print("--- Jogo da Forca ---")**

**print("Adivinhe a palavra!")**

**while erros < numero\_maximo\_erros:**

**print(f"\nPalavra: {mostrar\_palavra\_oculta(palavra\_secreta, letras\_corretas)}")**

**print(f"Letras tentadas: {letras\_tentadas}")**

**print(f"Erros restantes: {numero\_maximo\_erros - erros}")**

**letra = input("Digite uma letra: ").upper()**

**if letra in letras\_tentadas:**

**print(f"Você já tentou a letra '{letra}'. Tente outra.")**

**continue**

**letras\_tentadas.append(letra)**

**if letra in palavra\_secreta:**

**letras\_corretas.append(letra)**

**print(f"'{letra}' está na palavra!")**

**else:**

**erros += 1**

**print(f"'{letra}' não está na palavra.")**

**# Verifica se o jogador ganhou**

**if all(letra in letras\_corretas for letra in palavra\_secreta):**

**print(f"\n--- Fim de Jogo ---")**

**print(f"Parabéns, você ganhou! A palavra era '{palavra\_secreta}'.")**

**return**

**# Se o número máximo de erros for atingido**

**print(f"\n--- Fim de Jogo ---")**

**print(f"Você perdeu! A palavra era '{palavra\_secreta}'.")**

**# Executa o jogo**

**if \_\_name\_\_ == "\_\_main\_\_":**

**jogar()**

**import random**

**def escolher\_opcao():**

**"""Escolhe aleatoriamente entre pedra, papel ou tesoura para o computador."""**

**return random.choice(['pedra', 'papel', 'tesoura'])**

**def verificar\_vencedor(jogador, computador):**

**"""Verifica o resultado da partida entre jogador e computador."""**

**if jogador == computador:**

**return 'Empate'**

**elif (jogador == 'pedra' and computador == 'tesoura') or \**

**(jogador == 'papel' and computador == 'pedra') or \**

**(jogador == 'tesoura' and computador == 'papel'):**

**return 'Jogador'**

**else:**

**return 'Computador'**

**def jogar():**

**print("--- Jogo Pedra, Papel e Tesoura ---")**

**print("Opções válidas: pedra, papel, tesoura")**

**print("Digite 'sair' para encerrar o jogo.")**

**placar = {"Jogador": 0, "Computador": 0, "Empates": 0}**

**while True:**

**escolha\_jogador = input("\nDigite sua escolha: ").strip().lower()**

**if escolha\_jogador == 'sair':**

**print("\n--- Resultado Final ---")**

**print(f"Você venceu {placar['Jogador']} vezes.")**

**print(f"O computador venceu {placar['Computador']} vezes.")**

**print(f"Houve {placar['Empates']} empates.")**

**print("Encerrando jogo. Obrigado por jogar!")**

**break**

**if escolha\_jogador not in ['pedra', 'papel', 'tesoura']:**

**print("Escolha inválida! Tente novamente.")**

**continue**

**escolha\_computador = escolher\_opcao()**

**print(f"Computador escolheu: {escolha\_computador}")**

**resultado = verificar\_vencedor(escolha\_jogador, escolha\_computador)**

**if resultado == 'Empate':**

**print("Empate!")**

**placar['Empates'] += 1**

**elif resultado == 'Jogador':**

**print("Você venceu!")**

**placar['Jogador'] += 1**

**else:**

**print("Computador venceu!")**

**placar['Computador'] += 1**

**if \_\_name\_\_ == "\_\_main\_\_":**

**jogar()**

**import random**

**def mostrar\_tabuleiro(tabuleiro):**

**print("\n")**

**for i, linha in enumerate(tabuleiro):**

**print(" | ".join(linha))**

**if i < 2:**

**print("--+---+--")**

**print("\n")**

**def verificar\_vitoria(tabuleiro, marcador):**

**# Linhas**

**for linha in tabuleiro:**

**if all(pos == marcador for pos in linha):**

**return True**

**# Colunas**

**for col in range(3):**

**if all(tabuleiro[lin][col] == marcador for lin in range(3)):**

**return True**

**# Diagonais**

**if all(tabuleiro[i][i] == marcador for i in range(3)):**

**return True**

**if all(tabuleiro[i][2 - i] == marcador for i in range(3)):**

**return True**

**return False**

**def verificar\_empate(tabuleiro):**

**return all(pos != " " for linha in tabuleiro for pos in linha)**

**def solicitar\_jogada\_jogador(tabuleiro):**

**while True:**

**try:**

**pos = input("Sua jogada (linha,coluna) de 1 a 3 (ex: 2,3): ").strip()**

**linha, coluna = map(int, pos.split(','))**

**if linha < 1 or linha > 3 or coluna < 1 or coluna > 3:**

**print("Posição inválida. Use valores entre 1 e 3.")**

**continue**

**if tabuleiro[linha - 1][coluna - 1] != " ":**

**print("Posição já ocupada! Tente outra.")**

**continue**

**return linha - 1, coluna - 1**

**except:**

**print("Entrada inválida! Use o formato linha,coluna (ex: 1,3)")**

**def jogada\_computador(tabuleiro):**

**# Computador escolhe aleatoriamente uma posição vazia**

**vazias = [(i, j) for i in range(3) for j in range(3) if tabuleiro[i][j] == " "]**

**return random.choice(vazias) if vazias else None**

**def jogar():**

**print("--- Jogo da Velha: Jogador vs Computador ---")**

**tabuleiro = [[" "]\*3 for \_ in range(3)]**

**jogador = "X"**

**computador = "O"**

**while True:**

**mostrar\_tabuleiro(tabuleiro)**

**# Jogada do jogador**

**linha, coluna = solicitar\_jogada\_jogador(tabuleiro)**

**tabuleiro[linha][coluna] = jogador**

**if verificar\_vitoria(tabuleiro, jogador):**

**mostrar\_tabuleiro(tabuleiro)**

**print("Parabéns! Você venceu!")**

**break**

**if verificar\_empate(tabuleiro):**

**mostrar\_tabuleiro(tabuleiro)**

**print("Empate! Ninguém venceu.")**

**break**

**# Jogada do computador**

**jog\_comp = jogada\_computador(tabuleiro)**

**if jog\_comp is None:**

**mostrar\_tabuleiro(tabuleiro)**

**print("Empate! Ninguém venceu.")**

**break**

**tabuleiro[jog\_comp[0]][jog\_comp[1]] = computador**

**print(f"Computador jogou na posição: {jog\_comp[0]+1},{jog\_comp[1]+1}")**

**if verificar\_vitoria(tabuleiro, computador):**

**mostrar\_tabuleiro(tabuleiro)**

**print("Computador venceu! Tente novamente.")**

**break**

**if verificar\_empate(tabuleiro):**

**mostrar\_tabuleiro(tabuleiro)**

**print("Empate! Ninguém venceu.")**

**break**

**if \_\_name\_\_ == "\_\_main\_\_":**

**jogar()**

**import random**

**def criar\_tabuleiro():**

**# 'b' para peça preta, 'p' para peça branca, '.' para casa vazia, ' ' para casa clara (inválida)**

**tab = []**

**for linha in range(8):**

**row = []**

**for col in range(8):**

**if (linha + col) % 2 == 1:**

**if linha < 3:**

**row.append('b') # peças pretas (computador)**

**elif linha > 4:**

**row.append('p') # peças brancas (jogador)**

**else:**

**row.append('.')**

**else:**

**row.append(' ')**

**tab.append(row)**

**return tab**

**def mostrar\_tabuleiro(tab):**

**print("\n 0 1 2 3 4 5 6 7")**

**print(" -----------------")**

**for i, row in enumerate(tab):**

**print(f"{i} | {' '.join(row)}")**

**print()**

**def posicoes\_possiveis(tab, jogador):**

**# Retorna lista de movimentos possíveis (origem, destino) para jogador 'p' ou 'b'**

**direcao = -1 if jogador == 'p' else 1**

**capturas = []**

**simples = []**

**for r in range(8):**

**for c in range(8):**

**if tab[r][c].lower() == jogador:**

**# movimentos simples**

**for dc in [-1, 1]:**

**nr = r + direcao**

**nc = c + dc**

**if 0 <= nr < 8 and 0 <= nc < 8 and tab[nr][nc] == '.':**

**simples.append(((r,c),(nr,nc)))**

**# capturas**

**for dc in [-1, 1]:**

**nr = r + direcao**

**nc = c + dc**

**nr2 = r + 2\*direcao**

**nc2 = c + 2\*dc**

**if 0 <= nr2 < 8 and 0 <= nc2 < 8:**

**if tab[nr][nc].lower() != jogador and tab[nr][nc] != '.' and tab[nr][nc] != ' ' and tab[nr2][nc2] == '.':**

**capturas.append(((r,c),(nr2,nc2)))**

**return capturas if capturas else simples**

**def aplicar\_movimento(tab, origem, destino):**

**r0,c0 = origem**

**r1,c1 = destino**

**tab[r1][c1] = tab[r0][c0]**

**tab[r0][c0] = '.'**

**# verifica captura**

**if abs(r1 - r0) == 2:**

**rm = (r0 + r1)//2**

**cm = (c0 + c1)//2**

**tab[rm][cm] = '.'**

**# promoção a dama - não implementada para simplificar**

**def jogo\_terminado(tab):**

**pecas\_p = sum(row.count('p') + row.count('P') for row in tab)**

**pecas\_b = sum(row.count('b') + row.count('B') for row in tab)**

**return pecas\_p == 0 or pecas\_b == 0**

**def jogador\_move(tab):**

**movimentos = posicoes\_possiveis(tab, 'p')**

**if not movimentos:**

**return False**

**while True:**

**try:**

**mostrar\_tabuleiro(tab)**

**print("Suas peças: p")**

**origem = input("Digite posição da peça para mover (ex: 5 0): ")**

**destino = input("Digite posição destino (ex: 4 1): ")**

**r0,c0 = map(int, origem.strip().split())**

**r1,c1 = map(int, destino.strip().split())**

**if ((r0,c0),(r1,c1)) in movimentos:**

**aplicar\_movimento(tab, (r0,c0), (r1,c1))**

**return True**

**else:**

**print("Movimento inválido. Tente novamente.")**

**except Exception:**

**print("Entrada inválida. Use formato linha coluna, ex: 5 0")**

**def computador\_move(tab):**

**movimentos = posicoes\_possiveis(tab, 'b')**

**if not movimentos:**

**return False**

**movimento = random.choice(movimentos)**

**aplicar\_movimento(tab, movimento[0], movimento[1])**

**print(f"Computador moveu de {movimento[0]} para {movimento[1]}")**

**return True**

**def jogar\_damas():**

**print("--- Jogo de Damas simplificado contra computador ---")**

**tab = criar\_tabuleiro()**

**jogador\_turno = True # True = jogador, False = computador**

**while True:**

**if jogo\_terminado(tab):**

**mostrar\_tabuleiro(tab)**

**if jogador\_turno:**

**print("Fim de jogo! Computador venceu.")**

**else:**

**print("Fim de jogo! Você venceu!")**

**break**

**if jogador\_turno:**

**if not jogador\_move(tab):**

**mostrar\_tabuleiro(tab)**

**print("Você não tem movimentos válidos. Computador venceu!")**

**break**

**else:**

**if not computador\_move(tab):**

**mostrar\_tabuleiro(tab)**

**print("Computador não tem movimentos válidos. Você venceu!")**

**break**

**jogador\_turno = not jogador\_turno**

**if \_\_name\_\_ == "\_\_main\_\_":**

**jogar\_damas()**

**import os # Importa o módulo os para achar o caminho do arquivo**

**def ler\_dados\_vendas(nome\_arquivo):**

**"""Lê o arquivo de vendas e retorna uma lista de dicionários com os dados."""**

**vendas = []**

**with open(nome\_arquivo, "r", encoding="utf-8") as arquivo:**

**for linha in arquivo:**

**codigo, nome, quantidade, preco = linha.strip().split(";")**

**vendas.append({**

**"codigo": codigo,**

**"nome": nome,**

**"quantidade": int(quantidade),**

**"preco": float(preco)**

**})**

**return vendas**

**def processar\_vendas(dados\_vendas):**

**"""Processa os dados de vendas e retorna um dicionário consolidado."""**

**consolidado = {}**

**for venda in dados\_vendas:**

**codigo = venda["codigo"]**

**if codigo not in consolidado:**

**consolidado[codigo] = {**

**"nome": venda["nome"],**

**"quantidade\_total": 0,**

**"valor\_total": 0.0**

**}**

**consolidado[codigo]["quantidade\_total"] += venda["quantidade"]**

**consolidado[codigo]["valor\_total"] += venda["quantidade"] \* venda["preco"]**

**return consolidado**

**def gerar\_relatorio\_produto(dados\_consolidados, codigo\_produto):**

**"""Gera um relatório para um produto específico."""**

**if codigo\_produto in dados\_consolidados:**

**produto = dados\_consolidados[codigo\_produto]**

**print(f"--- Relatório Produto: {codigo\_produto} ---")**

**print(f"Nome: {produto['nome']}")**

**print(f"Quantidade Total Vendida: {produto['quantidade\_total']}")**

**print(f"Valor Total Vendido: R$ {produto['valor\_total']:.2f}")**

**else:**

**print(f"Produto {codigo\_produto} não encontrado.")**

**def gerar\_relatorio\_geral(dados\_consolidados):**

**"""Gera um relatório geral de vendas, ordenado pelo código do produto."""**

**print("--- Relatório Geral de Vendas ---")**

**for codigo, produto in sorted(dados\_consolidados.items()):**

**print(f"Produto: {codigo} - Nome: {produto['nome']} - Qtd: {produto['quantidade\_total']} - Total: R$ {produto['valor\_total']:.2f}")**

**# Programa principal**

**def main():**

**# Obtem o caminho absoluto do diretório onde está o script atual**

**diretorio\_script = os.path.dirname(os.path.abspath(\_\_file\_\_))**

**# Cria o caminho completo para o arquivo vendas.txt no mesmo diretório**

**nome\_arquivo = os.path.join(diretorio\_script, "vendas.txt")**

**dados\_vendas = ler\_dados\_vendas(nome\_arquivo)**

**dados\_consolidados = processar\_vendas(dados\_vendas)**

**print("Dados de vendas processados.")**

**while True:**

**codigo\_produto = input("\nDigite o código do produto para relatório (ou 'TODOS', ou 'SAIR'): ").strip().upper()**

**if codigo\_produto == "SAIR":**

**print("Encerrando programa.")**

**break**

**elif codigo\_produto == "TODOS":**

**gerar\_relatorio\_geral(dados\_consolidados)**

**else:**

**gerar\_relatorio\_produto(dados\_consolidados, codigo\_produto)**

**# Executar o programa principal**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

**def simular\_caixa\_eletronico():**

**saldo = 2000.00**

**print("Bem-vindo ao Caixa Eletrônico!")**

**print(f"Saldo inicial: R$ {saldo:.2f}\n")**

**while True:**

**print("""**

**Menu do Caixa Eletrônico**

**Digite a opção desejada:**

**1: Consultar Saldo**

**2: Realizar Saque**

**3: Realizar Depósito**

**4: Sair**

**""")**

**try:**

**opcao = int(input("Escolha uma opção: "))**

**except ValueError:**

**print("Opção inválida.")**

**continue**

**if opcao == 1:**

**print(f"Seu saldo atual é: R$ {saldo:.2f}")**

**elif opcao == 2:**

**valor\_saque = float(input("Digite o valor do saque: "))**

**if valor\_saque > 0:**

**if valor\_saque <= saldo:**

**saldo -= valor\_saque**

**print("Saque realizado com sucesso.")**

**else:**

**print("Saldo insuficiente.")**

**else:**

**print("Valor de saque inválido.")**

**elif opcao == 3:**

**valor\_deposito = float(input("Digite o valor do depósito: "))**

**if valor\_deposito > 0:**

**saldo += valor\_deposito**

**print("Depósito realizado com sucesso.")**

**else:**

**print("Valor de depósito inválido.")**

**elif opcao == 4:**

**print("Obrigado por usar nosso caixa eletrônico.")**

**break**

**else:**

**print("Opção inválida.")**

**# Chamar a função**

**simular\_caixa\_eletronico()**

**def celsius\_para\_fahrenheit(temp\_c):**

**"""Converte Celsius para Fahrenheit."""**

**return (temp\_c \* 9/5) + 32**

**def fahrenheit\_para\_celsius(temp\_f):**

**"""Converte Fahrenheit para Celsius."""**

**return (temp\_f - 32) \* 5/9**

**def gerenciar\_conversoes\_temperatura():**

**"""Gerencia o menu de conversão de temperaturas."""**

**while True:**

**print("""**

**\nMenu de Conversão de Temperatura:**

**\n[1]: Celsius para Fahrenheit**

**\n[2]: Fahrenheit para Celsius**

**\n[3]: Sair**

**""")**

**try:**

**opcao = int(input("Escolha uma opção: "))**

**except ValueError:**

**print("Opção inválida.")**

**continue**

**if opcao == 1:**

**temp\_c = float(input("Digite a temperatura em Celsius: "))**

**temp\_f = celsius\_para\_fahrenheit(temp\_c)**

**print(f"{temp\_c}°C é igual a {temp\_f}°F")**

**elif opcao == 2:**

**temp\_f = float(input("Digite a temperatura em Fahrenheit: "))**

**temp\_c = fahrenheit\_para\_celsius(temp\_f)**

**print(f"{temp\_f}°F é igual a {temp\_c}°C")**

**elif opcao == 3:**

**print("Encerrando o programa.")**

**break**

**else:**

**print("Opção inválida.")**

**# Chama a função principal**

**gerenciar\_conversoes\_temperatura()**

**def contar\_vogais\_consoantes(texto):**

**vogais = "aeiouAEIOU"**

**total\_vogais = 0**

**total\_consoantes = 0**

**for caractere in texto:**

**if caractere.isalpha(): # Verifica se é uma letra**

**if caractere in vogais:**

**total\_vogais += 1**

**else:**

**total\_consoantes += 1**

**# Imprime os resultados em vez de retornar**

**print(f"Vogais: {total\_vogais}, Consoantes: {total\_consoantes}")**

**# Exemplo de uso**

**contar\_vogais\_consoantes("Ola Programador!")**

**contar\_vogais\_consoantes("Python 3.12 eh Legal :)")**

**def contar\_vogais\_consoantes(texto):**

**vogais = "aeiouAEIOU"**

**total\_vogais = 0**

**total\_consoantes = 0**

**for caractere in texto:**

**if caractere.isalpha(): # Verifica se é uma letra**

**if caractere in vogais:**

**total\_vogais += 1**

**else:**

**total\_consoantes += 1**

**return total\_vogais, total\_consoantes**

**resultado1 = contar\_vogais\_consoantes("Ola Programador!")**

**print(f"Vogais: {resultado1[0]}, Consoantes: {resultado1[1]}")**

**resultado2 = contar\_vogais\_consoantes("Python 3.12 eh Legal :)")**

**print(f"Vogais: {resultado2[0]}, Consoantes: {resultado2[1]}")**

**saldo\_conta = 500**

**def realizar\_compra(valor\_compra):**

**taxa\_servico = 10**

**# saldo\_conta = saldo\_conta - valor\_compra - taxa\_servico # Linha A (comentada)**

**novo\_saldo\_local = saldo\_conta - valor\_compra - taxa\_servico # Linha B**

**if novo\_saldo\_local >= 0:**

**print(f"Compra de R${valor\_compra} aprovada. Saldo (local) após compra: R${novo\_saldo\_local}")**

**# saldo\_conta = novo\_saldo\_local # Linha C (comentada)**

**return True**

**else:**

**print(f"Compra de R${valor\_compra} negada. Saldo insuficiente.")**

**return False**

**print(f"Saldo inicial: R${saldo\_conta}")**

**realizar\_compra(100)**

**print(f"Saldo após primeira tentativa: R${saldo\_conta}")**

**realizar\_compra(400)**

**print(f"Saldo final: R${saldo\_conta}")**

**def eh\_primo(numero):**

**if numero <= 1:**

**return False # 1 e números negativos não são primos**

**# Verifica se o número é divisível por qualquer número de 2 até o número - 1**

**for i in range(2, numero):**

**if numero % i == 0:**

**return False # O número é divisível por i, portanto não é primo**

**return True # O número é primo**

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

**if eh\_primo(numero\_usuario):**

**print(f"{numero\_usuario} é primo.")**

**else:**

**print(f"{numero\_usuario} não é primo.")**

**def calcular\_area\_retangulo(base, altura):**

**return base \* altura**

**base = float(input("Digite a base do retângulo: "))**

**altura = float(input("Digite a altura do retângulo: "))**

**# Chama a função com os valores informados**

**area = calcular\_area\_retangulo(base, altura)**

**print(f"A área do retângulo com base {base} e altura {altura} é: {area}")**

**senha\_correta = "python123"**

**senha\_digitada = ""**

**while senha\_digitada != senha\_correta: # Loop que continua até a senha correta ser digitada**

**senha\_digitada = input("Digite a senha: ")**

**if senha\_digitada != senha\_correta:**

**print("Senha incorreta. Tente novamente.")**

**print("Acesso concedido!")**

**soma = 0**

**for numero in range(1, 50+1):**

**if numero % 2 == 0: # Verifica se o número é par**

**soma += numero # Adiciona o número par à soma**

**print(f"A soma dos números pares entre 1 e 50 é: {soma}")**

**numero1 = int(input("Digite o primeiro número: "))**

**numero2 = int(input("Digite o segundo número: "))**

**print("-"\*20)**

**print("ENTRADA".center(20))**

**print("-"\*20)**

**print(f"Primeiro número: {numero1}")**

**print(f"Segundo número : {numero2}")**

**print("-"\*20)**

**print("SAÍDA".center(20))**

**print("-"\*20)**

**soma = numero1 + numero2**

**subtracao = numero1 - numero2**

**multiplicacao = numero1 \* numero2**

**print(f"Soma: {soma}")**

**print(f"Subtração: {subtracao}")**

**print(f"Multiplicação: {multiplicacao}")**

**if numero2 != 0:**

**divisao = numero1 / numero2**

**print(f"Divisão: {divisao:.2f}")**

**else:**

**print("Divisão por zero não permitida")**

**print("-"\*20)**

**nome = input("Digite seu nome: ")**

**idade = int(input("Digite sua idade: "))**

**if idade >= 18:**

**print(f"{nome} é maior de idade.")**

**else:**

**print(f"{nome} é menor de idade.")**

**def menu():**

**print("\nMenu:")**

**print("1. Incluir cidade")**

**print("2. Excluir cidade")**

**print("3. Consultar cidade")**

**print("4. Listar cidades")**

**print("5. Backup")**

**print("6. Restore")**

**print("7. Sair")**

**return input("Escolha uma opção: ")**

**def incluir\_cidade(cidades):**

**while True:**

**cidade = input("Digite o nome da cidade para incluir: ")**

**if cidade not in cidades:**

**cidades.append(cidade)**

**print(f"{cidade} foi incluída na lista.")**

**else:**

**print(f"{cidade} já está na lista.")**

**continuar = input("Deseja incluir outra cidade? (s/n): ").strip().lower()**

**if continuar != 's':**

**break**

**def excluir\_cidade(cidades):**

**while True:**

**cidade = input("Digite o nome da cidade para excluir: ")**

**if cidade in cidades:**

**cidades.remove(cidade)**

**print(f"{cidade} foi removida da lista.")**

**else:**

**print(f"{cidade} não está na lista.")**

**continuar = input("Deseja excluir outra cidade? (s/n): ").strip().lower()**

**if continuar != 's':**

**break**

**def consultar\_cidade(cidades):**

**cidade = input("Digite o nome da cidade para consultar: ")**

**if cidade in cidades:**

**print(f"{cidade} está na lista.")**

**else:**

**print(f"{cidade} não está na lista.")**

**def listar\_cidades(cidades):**

**if cidades:**

**print("Lista de cidades:")**

**for cidade in cidades:**

**print(f"- {cidade}")**

**else:**

**print("A lista de cidades está vazia.")**

**def backup\_cidades(cidades):**

**if cidades:**

**try:**

**with open("backup\_cidades.txt", "w") as arquivo:**

**for cidade in cidades:**

**arquivo.write(f"{cidade}\n")**

**print("Backup realizado com sucesso! As cidades foram salvas no arquivo 'backup\_cidades.txt'.")**

**except Exception as e:**

**print(f"Ocorreu um erro ao tentar realizar o backup: {e}")**

**else:**

**print("A lista de cidades está vazia. Não há nada para fazer backup.")**

**def restore\_cidades(cidades):**

**try:**

**with open("backup\_cidades.txt", "r") as arquivo:**

**cidades\_restauradas = arquivo.readlines()**

**cidades.clear() # Limpa a lista atual antes de restaurar**

**cidades.extend([cidade.strip() for cidade in cidades\_restauradas]) # Adiciona as cidades restauradas**

**print("Restore realizado com sucesso! As cidades foram restauradas da lista de backup.")**

**except FileNotFoundError:**

**print("Nenhum arquivo de backup encontrado. Certifique-se de que o backup foi realizado antes.")**

**except Exception as e:**

**print(f"Ocorreu um erro ao tentar realizar o restore: {e}")**

**def main():**

**cidades = []**

**while True:**

**opcao = menu()**

**if opcao == "1":**

**incluir\_cidade(cidades)**

**elif opcao == "2":**

**excluir\_cidade(cidades)**

**elif opcao == "3":**

**consultar\_cidade(cidades)**

**elif opcao == "4":**

**listar\_cidades(cidades)**

**elif opcao == "5":**

**backup\_cidades(cidades)**

**elif opcao == "6":**

**restore\_cidades(cidades)**

**elif opcao == "7":**

**print("Saindo do programa. Até mais!")**

**break**

**else:**

**print("Opção inválida. Tente novamente.")**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

**menu = """**

**SISTEMA BANCARIO**

**=====================**

**[d] Deposito**

**[s] Saque**

**[e] Extrato**

**[l] Log de erros**

**[q] Sair**

**=====================**

**by jeds**

**=> """**

**saldo = float(0)**

**valor = float(0)**

**limite = float(500)**

**extrato = ""**

**numero\_saques = 0**

**LIMITE\_SAQUES = 3**

**log\_erros = ""**

**while True:**

**opcao = input(menu)**

**if opcao == "d":**

**print("DEPOSITO")**

**valor = input("Qual valor você deseja depositar?: "))**

**if valor > 0:**

**saldo += valor**

**extrato += f"Depósito (+): R$ {valor:10.2f}\n"**

**print(f"Você acabou de depositar R$ {valor:10.2f} no seu saldo.\n")**

**else:**

**print("Informe um valor válido!")**

**log\_erros += f"Informado um valor inválido para depósito. Valor: {valor:10.2f}\n"**

**elif opcao == "s":**

**print("SAQUE")**

**valor = (input("Qual o valor que você deseja sacar?: "))**

**excedeu\_saldo = valor > saldo**

**excedeu\_limite = valor > limite**

**excedeu\_saques = numero\_saques >= LIMITE\_SAQUES**

**if excedeu\_saldo:**

**print("Operação falhou! Saldo insuficiente!")**

**log\_erros += f"Operação falhou! Saldo insuficiente! Valor : {valor:10.2f}\n"**

**elif excedeu\_limite:**

**print("Operação falhou! Valor do saque excede o limite!")**

**log\_erros += f"Operação falhou! Valor do saque excede o limite! Valor: {valor:10.2f}\n"**

**elif excedeu\_saques:**

**print("Operação falhou! Você excedeu o número de saques diários!")**

**log\_erros += f"Operação falhou! Você excedeu o número de saques! Quantidade: {numero\_saques}\n"**

**elif valor > 0:**

**saldo -= valor**

**extrato +=f"Saque (-): R$ {valor:10.2f}\n"**

**print(f"Você acabou de sacar R$ {valor:10.2f} do seu saldo.\n")**

**numero\_saques += 1**

**else:**

**print("Operação falhou! O valor do saque deve ser maior que zero!")**

**log\_erros += f"Operação falhou! O valor do saque deve ser maior que zero! Valor: {valor:10.2f}\n"**

**elif opcao == "e":**

**print("\n================= EXTRATO =================")**

**print("Não foram realizadas movimentações." if not extrato else extrato)**

**print(f"\nSaldo Atual: R$ {saldo:10.2f}")**

**print("============================================")**

**elif opcao == "l":**

**print("\n============================= ERROS ==============================")**

**print("Não foram registrados erros." if not log\_erros else log\_erros)**

**print("=================================================================")**

**elif opcao == "q":**

**print("Saindo do sistema...")**

**break**

**else:**

**print("Opção inválida. Tente novamente!")**

**import json**

**import os**

**class Produto:**

**def \_\_init\_\_(self, codigo, nome, quantidade, valor, categoria=None):**

**self.codigo = codigo**

**self.nome = nome**

**self.quantidade = quantidade**

**self.valor = valor**

**self.categoria = categoria**

**def \_\_repr\_\_(self):**

**return f"{self.codigo}: {self.nome} - {self.quantidade} unidades - R${self.valor:.2f} - Categoria: {self.categoria}"**

**class ControleEstoque:**

**def \_\_init\_\_(self):**

**self.estoque = {}**

**self.carregar\_estoque()**

**def carregar\_estoque(self):**

**try:**

**with open("estoque.json", "r") as file:**

**estoque\_carregado = json.load(file)**

**self.estoque = {codigo: Produto(**

**\*\*dados) for codigo, dados in estoque\_carregado.items()}**

**print("Estoque carregado com sucesso!")**

**except FileNotFoundError:**

**print("Arquivo de estoque não encontrado. Um novo será criado.")**

**def salvar\_estoque(self):**

**with open("estoque.json", "w") as file:**

**json.dump({codigo: produto.\_\_dict\_\_ for codigo,**

**produto in self.estoque.items()}, file)**

**print("Estoque salvo com sucesso!")**

**def incluir\_produto(self):**

**codigo = input("Digite o código do produto: ")**

**if codigo in self.estoque:**

**print("Este código já existe.")**

**opcao = input(**

**"Deseja acrescentar itens ao produto existente? (s/n): ")**

**if opcao.lower() == 's':**

**quantidade = int(**

**input("Digite a quantidade de itens a acrescentar: "))**

**self.estoque[codigo].quantidade += quantidade**

**print("Quantidade atualizada com sucesso!")**

**else:**

**print("Operação cancelada.")**

**else:**

**nome = input("Digite o nome do produto: ")**

**quantidade = int(input("Digite a quantidade do produto: "))**

**valor = float(input("Digite o valor do produto: "))**

**categoria = input("Digite a categoria do produto (opcional): ")**

**produto = Produto(codigo, nome, quantidade, valor, categoria)**

**self.estoque[codigo] = produto**

**print("Produto incluído com sucesso!")**

**self.salvar\_estoque()**

**def excluir\_produto(self):**

**codigo = input("Digite o código do produto que deseja excluir: ")**

**if codigo in self.estoque:**

**del self.estoque[codigo]**

**print("Produto excluído com sucesso!")**

**self.salvar\_estoque()**

**else:**

**print("Produto não encontrado.")**

**def alterar\_produto(self):**

**codigo = input("Digite o código do produto que deseja alterar: ")**

**if codigo in self.estoque:**

**nome = input("Digite o novo nome do produto: ")**

**quantidade = int(input("Digite a nova quantidade do produto: "))**

**valor = float(input("Digite o novo valor do produto: "))**

**categoria = input("Digite a nova categoria do produto (opcional): ")**

**produto = self.estoque[codigo]**

**produto.nome = nome**

**produto.quantidade = quantidade**

**produto.valor = valor**

**produto.categoria = categoria**

**print("Produto alterado com sucesso!")**

**self.salvar\_estoque()**

**else:**

**print("Produto não encontrado.")**

**def consultar\_produto(self):**

**codigo = input("Digite o código do produto que deseja consultar: ")**

**if codigo in self.estoque:**

**print(self.estoque[codigo])**

**else:**

**print("Produto não encontrado.")**

**def listar\_produtos(self):**

**if not self.estoque:**

**print("Não há produtos no estoque.")**

**else:**

**print(**

**f"{'CÓDIGO':>10} {'PRODUTO':<20} {'UNIDADES':>10} {'VALOR':>12} {'CATEGORIA':<15} {'VALOR TOTAL':>14}")**

**print("="\*85)**

**total\_quantidade = 0**

**total\_valor\_geral = 0.0**

**for produto in self.estoque.values():**

**valor\_total = produto.quantidade \* produto.valor**

**print(**

**f"{produto.codigo:>10} {produto.nome:<20} {produto.quantidade:>10} R${produto.valor:>10.2f} {produto.categoria:<15} R${valor\_total:>12.2f}")**

**total\_quantidade += produto.quantidade**

**total\_valor\_geral += valor\_total**

**print("="\*85)**

**print(**

**f"{'TOTAL':<31} {total\_quantidade:>10} {'':<15} {'':<12} R${total\_valor\_geral:>12.2f}")**

**def imprimir\_lista\_produtos(self):**

**if not self.estoque:**

**print("Não há produtos no estoque.")**

**else:**

**with open("lista\_produtos.txt", "w") as file:**

**file.write(**

**f"{'CÓDIGO':>10} {'PRODUTO':<20} {'UNIDADES':>10} {'VALOR':>12} {'CATEGORIA':<15} {'VALOR TOTAL':>14}\n")**

**file.write("="\*85 + "\n")**

**total\_quantidade = 0**

**total\_valor\_geral = 0.0**

**for produto in self.estoque.values():**

**valor\_total = produto.quantidade \* produto.valor**

**file.write(**

**f"{produto.codigo:>10} {produto.nome:<20} {produto.quantidade:>10} R${produto.valor:>10.2f} {produto.categoria:<15} R${valor\_total:>12.2f}\n")**

**total\_quantidade += produto.quantidade**

**total\_valor\_geral += valor\_total**

**file.write("="\*85 + "\n")**

**file.write(**

**f"{'TOTAL':<31} {total\_quantidade:>10} {'':<15} {'':<12} R${total\_valor\_geral:>12.2f}\n")**

**os.system("notepad /p lista\_produtos.txt")**

**print("Lista de produtos enviada para impressão.")**

**def mostrar\_total(self):**

**total\_quantidade = sum(**

**produto.quantidade for produto in self.estoque.values()**

**)**

**total\_valor = sum(**

**produto.quantidade \* produto.valor for produto in self.estoque.values()**

**)**

**print(f"Total de produtos em quantidade: {total\_quantidade}")**

**print(f"Total de produtos em valor: R${total\_valor:.2f}")**

**def backup(self):**

**with open("estoque\_backup.json", "w") as file:**

**json.dump(**

**{codigo: produto.\_\_dict\_\_ for codigo, produto in self.estoque.items()},**

**file,**

**)**

**print("Backup realizado com sucesso!")**

**def restaurar(self):**

**try:**

**with open("estoque\_backup.json", "r") as file:**

**estoque\_backup = json.load(file)**

**self.estoque = {**

**codigo: Produto(\*\*dados) for codigo, dados in estoque\_backup.items()**

**}**

**print("Restauração realizada com sucesso!")**

**self.salvar\_estoque()**

**except FileNotFoundError:**

**print("Arquivo de backup não encontrado.")**

**def menu(self):**

**while True:**

**print("\nMENU CONTROLE DE ESTOQUE")**

**print("1. INCLUIR PRODUTO")**

**print("2. EXCLUIR PRODUTO")**

**print("3. ALTERAR PRODUTO")**

**print("4. CONSULTAR PRODUTO")**

**print("5. LISTAR PRODUTOS")**

**print("6. IMPRIMIR LISTA DE PRODUTOS")**

**print("7. MOSTRAR TOTAL DE PRODUTOS")**

**print("8. BACKUP DO ESTOQUE")**

**print("9. RESTAURAR ESTOQUE")**

**print("10. SAIR")**

**opcao = input("Escolha uma opção: ")**

**if opcao == "1":**

**self.incluir\_produto()**

**elif opcao == "2":**

**self.excluir\_produto()**

**elif opcao == "3":**

**self.alterar\_produto()**

**elif opcao == "4":**

**self.consultar\_produto()**

**elif opcao == "5":**

**self.listar\_produtos()**

**elif opcao == "6":**

**self.imprimir\_lista\_produtos()**

**elif opcao == "7":**

**self.mostrar\_total()**

**elif opcao == "8":**

**self.backup()**

**elif opcao == "9":**

**self.restaurar()**

**elif opcao == "10":**

**print("Saindo...")**

**break**

**else:**

**print("Opção inválida. Tente novamente.")**

**if \_\_name\_\_ == "\_\_main\_\_":**

**controle\_estoque = ControleEstoque()**

**controle\_estoque.menu()**

**1 – Comprimir Sequencia**

**2 – Criptografar e Descriptografar**

**3 – Modificar Lista**

**4 – Dados Vendas com Criação**

**7 – Alterar Dados**

**9 – Ler Diretórios**

**11 – Dados Alunos com Criação**

**14 – Jogo da Forca**

**16 – Jogo Papel**

**18 – Jogo da Velha**

**21 – Jogo de Damas**

**25 – Dados Vendas**

**27 – Caixa Eletrônico Simples**

**29 – Conversão C -> F**

**30 – Contar Vogais e Consoantes**

**31 – Realizar Compra / Primo**

**32 – Área Retângulo / Senha / Pares**

**33 – Soma números / Entrada Simples**

**34 – Cadastro de Cidades**

**37 – Sistema Bancário**

**40 – Controle de Estoque com json**