Relatório: Implementação de Estruturas de Matrizes em Python

1. Estruturas de Dados Empregadas

A presente implementação fundamenta-se em uma hierarquia de classes, visando a representação de distintos tipos de matrizes:

- `Matriz_Geral`: Classe base para matrizes genéricas de ordem m × n.
- `Quadrada`: Subclasse de `Matriz_Geral`, especializada em matrizes de ordem n × n.
- `Triangular_Inf` e `Triangular_Sup`: Classes designadas para matrizes triangulares inferiores e superiores, respectivamente.
- `Diagonal`: Subclasse de `Quadrada`, dedicada a matrizes diagonais.

2. Atributos Fundamentais:

- `m`, `n`: Dimensões da matriz, indicando o número de linhas e colunas.
- `valores`: Estrutura de lista de listas, destinada ao armazenamento dos elementos matriciais.

3. Organização Modular:

- `matrizes.py`:
  - Contém as definições das classes de matrizes e as operações matemáticas associadas (+, -, *, @, transposição).
  - Inclui funções auxiliares, tais como `read_matriz`, `write_matriz`, `delete_matriz` e `label_matriz`.
- `menu.py`:
  - Gerencia a interface com o usuário através da classe `MatrizManager`.
  - Oferece operações interativas, como inserção, alteração, remoção, listagem e backup de matrizes.

4. Detalhamento das Rotinas e Funções:

Operações Fundamentais (Classe `Matriz_Geral`)

Métodos	Descrição
`__init__`	Validação das dimensões e valores iniciais; verificação do tipo numérico.
`set_Valores`	Atualização dos valores da matriz com verificação de dimensões.
`verificar_Tipo_Matriz`	Classificação da matriz (diagonal,

	triangular, quadrada, geral).
`__add__` & `__sub__`	Operações de soma/subtração de matrizes (compatibilidade verificada).
`__matmul__`	Multiplicação de matrizes (otimizada para tipos específicos).
`__mul__` & `__rmul__`	Multiplicação por um escalar.
`Transposicao`	Retorna a matriz transposta.

5. Métodos Especializados

- `Quadrada`:
  - `Calcular_Traço`: Calcula a soma dos elementos na diagonal principal.

- `Triangular` (Inferior/Superior):
  - `Calcular_Determinante`: Calcula o determinante (produto dos elementos da diagonal - O(n)).

- `Diagonal`:
  - Operações otimizadas (ex: multiplicação usando apenas elementos não nulos).

6. Funções de Arquivo

Funções	**Descrição**
`read_matriz`	Leitura de matriz de um arquivo (valores separados por espaço).
`write_matriz`	Salva a matriz em um arquivo com formatação adequada.
`delete_matriz`	Remove uma matriz específica do arquivo.

7. Análise de Complexidade (Tempo e Espaço)

Operação	Tempo	Espaço	Explicação
`__add__`/`__sub__`	O(m × n)	O(m × n)	Percorre todos os elementos da matriz.
`__matmul__`	O(m × n…	O(m × p)	Requer um triplo loop para a multiplicação.
`__mul__` (escalar)	O(m × n)	O(m × n)	Percorre todos os elementos.
`Transposicao`	O(m × n)	O(m × n)	Cria uma nova matriz transposta.
`Calcular_Traço`	O(n)	O(1)	Itera apenas sobre a diagonal principal.
`Calcular_Determinante`	O(n)	O(1)	Calcula o produto dos elementos da diagonal.

8. Conclusão

A implementação detalhada do sistema revela um domínio avançado sobre os princípios da programação orientada a objetos, particularmente no que tange à herança e ao polimorfismo. Estas técnicas foram aplicadas com o objetivo claro de otimizar a execução de operações matriciais, adaptando o comportamento do sistema de acordo com o tipo específico da matriz em questão. Tal abordagem não apenas aumenta a eficiência do código, mas também demonstra uma compreensão profunda de boas práticas de design de software, promovendo a reutilização de código e a flexibilidade do sistema.