FGV EMAp
ESCOLA DE MATEMÁTICA APLICADA

Prova 3 de Engenharia de Software
13 de Outubro de 2025

Nome: _____

Instruções Gerais

- Todas as questões da prova devem ser organizadas em **dois arquivos apenas**:
  1. Um módulo denominado `funcoes.py`, no qual deverão ser implementadas todas as funções desenvolvidas ao longo da prova;
  2. Um arquivo `main.py`, que atuará como **driver code**, responsável por importar o módulo `funcoes.py` e demonstrar o funcionamento das funções nele implementadas.
- O código deverá apresentar-se **claro, devidamente organizado e adequadamente documentado**. Apenas serão aceitas soluções que evidenciem legibilidade, coerência estrutural e rigor metodológico.

Questão 1 (5 pontos) Implemente dois decoradores para realizar o **cache (armazenamento) de resultados de funções**, evitando novas invocações para se obter resultados conhecidos. Considere apenas os argumentos posicionais `*args` para o registro no cache, e **ignore** `**kwargs`. *Observação:* a variável `args`, recebida pelo decorador, é uma `tuple` e pode ser utilizada diretamente como chave de dicionários e conjuntos.

(a) Implemente o decorador `cache_resultados` que armazene os resultados de chamadas anteriores de uma função. Se a função for chamada novamente com os mesmos argumentos, o resultado deve ser retornado do cache sem executar novamente a função original.

(b) Implemente `cache_limitado(max_tamanho)`, com comportamento análogo ao item (a), porém com um **limite máximo de entradas**. Quando o número de entradas no cache atingir `max_tamanho`, **o registro mais antigo deve ser eliminado para que o registro novo seja adicionado.**

Exemplo:

```
Chamada: soma(42, 13)
Resultado calculado = 55
=> valor armazenado no cache

Chamada: soma(42, 13)
=> resultado retornado do cache (55), sem recalcular
```

Questão 2 (2 pontos)

===

Questão 4 (2 pontos) — Adapter: uniformizando interfaces distintas com contrato explícito

Contexto
O cliente do sistema consome objetos que expõem a interface única executar(entrada: str) -> str. Dois serviços legados não seguem essa interface:

- ServicoTextoA.processar(texto: str) -> str
- ServicoTextoB.run(payload: dict) -> dict, onde payload deve ser {"data": "<texto>"} e a resposta vem como {"resultado": "<string>"}

Objetivo
Tornar ambos utilizáveis por um mesmo cliente, sem alterar os serviços legados, por meio do padrão Adapter, exigindo um contrato explícito (ABC) e normalização de entrada/saída.

Tarefas

1. Contrato (ABC) do cliente
   Defina uma classe abstrata Executavel (via abc.ABC) com o método abstrato executar(entrada: str) -> str. O objetivo é explicitar o contrato esperado pelo cliente, que ambos os adapters devem implementar.
2. Adapters por composição com normalização
   Implemente duas classes que herdam de Executavel e recebem a instância do serviço legado no construtor:
   - AdapterA(servico_a: ServicoTextoA, pre = None, pos = None)
     Regras de executar(entrada):
     (i) aplicar pre(entrada) se pre for uma função; caso contrário, usar a própria entrada;
     (ii) delegar para servico_a.processar(entrada_normalizada);
     (iii) aplicar pos(resultado) se pos for uma função; caso contrário, retornar o resultado como está.
   - AdapterB(servico_b: ServicoTextoB, pre = None, pos = None)
     Regras de executar(entrada):
     (i) aplicar pre(entrada) se existir; caso contrário, usar a própria entrada;
     (ii) montar {"data": entrada_normalizada} e chamar servico_b.run(...);
     (iii) se a resposta não for um dicionário, não contiver a chave "resultado" ou o valor não for string, retornar "" (string vazia);
     (iv) se houver pos, aplicar ao valor de "resultado" antes de retornar.

Regras de robustez

- Se entrada for None, tratar como "" (string vazia).

- Se entrada não for string, converter via str(entrada) antes de continuar.
- O uso de pre/pos é opcional e serve para normalizações simples (por exemplo, str.strip, str.upper).

Exemplo esperado

```
from abc import ABC, abstractmethod
class ServicoTextoA:
    def processar(self, texto: str) -> str:
        return f"[A]{texto}"

class ServicoTextoB:
    def run(self, payload: dict) -> dict:
        data = payload.get("data", "")
        return {"resultado": str(data)[::-1]}

a = AdapterA(ServicoTextoA(), pre=str.strip, pos=str.upper)
b = AdapterB(ServicoTextoB())
a.executar("  ola ")
b.executar("abc")
```