Trabalho — Paradigmas de Programação em Python (3 questões, 3,0 pts)

- **Instruções**
- Responda no papel, letra legível.
- Mostre o raciocínio (passos, tabelas de rastreio, justificativas).
- Pseudocódigo ou Python simplificado são aceitos quando pedido.

Q1) Paradigma **Estruturado** — algoritmo + dry run (1,0 pt)

Temos leituras de água (m³) em um mês:

```
[8.5, 7.2, 9.1, 11.0, 10.4, 6.9, 8.9]
```

1. Escreva um **algoritmo estruturado** (passo a passo ou pseudocódigo) para calcular:
   - média,
   - mínimo e máximo,
   - contagem de valores **acima** da média.
     (Não use ordenação.)
2. Faça o **dry run** do seu algoritmo sobre a lista acima mostrando uma **tabela de rastreio** com as variáveis principais (por exemplo: `v`, `soma`, `min`, `max`, `acima_media?` — você pode calcular a média ao final e então fazer uma segunda passada curta para contar os “acima da média”, ou explicar como faria em uma única passada).

Rubrica (1,0):

- Algoritmo claro, sem lacunas (0,4)
- Cálculos corretos (0,4)
- Tabela de rastreio legível (0,2)

Q2) Paradigma **Funcional** — reescrita pura +

resultado (1,0 pt)

Dado o laço imperativo:

```
resultado = []
for x in leituras:
    if 8 <= x <= 11:
        resultado.append(round(x * x, 2))
```

a) Reescreva em **estilo funcional** de duas formas, **sem efeitos colaterais**:

- (A) **compreensão de listas**
- (B) `filter(...) + map(...)` (pode usar `lambda`)

b) Para `leituras = [7, 8, 8.5, 11, 12]`, calcule **à mão** o `resultado` final da sua versão funcional (mostre as etapas: filtragem → mapeamento → arredondamento).

c) Em **uma frase**, explique por que sua função é **pura** (dica: ausência de estado compartilhado e de mutação observável).

Rubrica (1,0):

- Reescrita com compreensão correta (0,35)
- Reescrita com `filter` + `map` correta (0,35)
- Cálculo manual correto (0,2)
- Justificativa de pureza objetiva (0,1)

Q3) Paradigma **POO** — mini-modelagem com polimorfismo (1,0 pt)

No papel, modele uma solução para calcular a **cobrança do mês** por unidade de condomínio.

- **Requisitos mínimos**
- Classe `Medicao` com: `mes: str`, `agua_m3: float`, `energia_kwh: float` (**sem negativos**).
- Classe `Unidade` com: `numero: str` e lista de medições; método `adicionar_medicao(m)`.
- Classe `CriterioCobranca` (base) com método `calcular(unidade, mes) -> valor`.
- Duas subclasses:
  - `CobrancaProporcional`: cobra por consumo do mês (explique a fórmula).

    - `CobrancaRateioFixo` : divide igualmente um valor total entre as unidades ativas no mês (explique a regra).
- Classe `Fatura(mes, unidades, criterio)` com método `total_por_unidade() -> dict` .
- **Tarefas**

1. Desenhe/esboce as **classes** com **atributos** e **métodos** essenciais (assinaturas bastam).
2. Em **duas frases**, aponte **onde há encapsulamento** (ex.: validações em `Medicao`) e **onde ocorre polimorfismo** (chamada a `criterio.calcular(...)`).
3. Liste **duas validações** que devem lançar erro (ex.: consumo negativo; `mes` com formato inválido).

Rubrica (1,0):

- Estrutura das classes e relações coerentes (0,5)
- Explicação de encapsulamento e polimorfismo (0,3)
- Validações pertinentes (0,2)