

## PERGUNTA 04 - CP1 MMC

Pergunta 4 (3 pontos) - Pedir para o chatGPT (ou outra LLM) para refazer as questões 2 e 3. Faça o prompt da forma que você ache mais adequada. Na resposta, você deve incluir o prompt utilizado e também as linhas de código geradas. Execute esse output. Ele está correto? Avalie

### SOBRE A PERGUNTA 02

prompt: (ANEXEI A FOTO DA TABELA) Fiz uma atividade de Matematica com python baseada nesta pergunta: Sobre o mesmo PDF, escolher 3 erros algébricos comuns (Common Algebraic Errors) e verificá-los utilizando Python.

e essa foi a minha atividade:

(COLOQUEI MEU CÓDIGO AQUI)

analise minha atividade e a refaça como voce achar melhor

chat: Gostei bastante da sua atividade! 🚀

Você conseguiu traduzir bem os **erros algébricos comuns** mostrados no PDF para Python.

Vou te dar uma análise e depois propor uma versão refeita, mais organizada e clara:

---

### Análise da sua atividade

#### 1. Divisão por zero

- Você tentou calcular  $2/0$ , mas isso gera `ZeroDivisionError`. Bom que você destacou que é indefinida, mas seria melhor **tratar o erro com `try/except`** para mostrar a mensagem sem quebrar o programa.

#### 2. Potenciação de números negativos

- O exemplo está ótimo:  $(-3)**2$  dá 9, mas  $-3**2$  dá -9. Você mostrou a diferença com e sem parênteses, o que é exatamente o erro que muitos cometem.

#### 3. Potência de potência

- Você demonstrou a diferença entre **somar expoentes** (errado) e **multiplicar expoentes** (certo). Isso ficou bom!

- Mas poderia simplificar a visualização mostrando direto o resultado de cada caso.

---

## ✓ Versão refeita e melhorada

Aqui está uma versão mais clara, organizada e com tratamento de erro no primeiro caso:

```
print("\n1) Division by zero is undefined! -----")

num = 2
den = 0

try:
    fracao = num / den
except ZeroDivisionError:
    print(f"Tentando calcular {num}/{den} -> Erro: Divisão por zero é indefinida!\n")

print("\n2) Potenciação de números negativos -----")

pot_parenteses = (-3)**2    # correto
pot_sem_parenteses = -3**2  # errado por falta de parênteses

print(f"(-3)**2 = {pot_parenteses} (correto)")
print(f"-3**2 = {pot_sem_parenteses} (cuidado, o - não está elevado!)\n")

print("\n3) Potência de potência -----")

base = 2
a = 3
b = 4

# erro comum: somar expoentes
resultado_errado = base**(a+b)

# correto: multiplicar expoentes
resultado_correto = (base**a)**b

print(f"(2**3)**4 = {resultado_correto} (correto, pois 3*4 = 12 → 2**12 = {resultado_correto})")
print(f"2**(3+4) = {resultado_errado} (errado, pois soma expoentes ao invés de multiplicar)\n")
```

---

## 🔨 Principais melhorias que fiz

- Tratei a divisão por zero com `try/except` para mostrar a mensagem sem quebrar o código.
- Deixei a explicação junto com os prints, para ficar autoexplicativo.
- Evitei variáveis desnecessárias no último exemplo, mostrando direto a comparação entre o erro e o correto.

ele está correto?

- Pelo meu entendimento, o chat refez minha atividade corretamente. Achei interessante como ele mostrou o erro da divisão de um numerador por 0, não sabia como fazer isso então só deixei que o erro que aparece no terminal comprovasse esse erro algébrico.

### SOBRE A PERGUNTA 03

**prompt:** (ANEXEI A FOTO DA TABELA) Agora sobre outra atividade, fiz com base nesta pergunta: Pergunta 3 (3 pontos) - Verifique essas propriedades utilizando numpy

(numérico). Gere vetores numéricos e verifique as igualdades do slide anterior. Definir vetores arbitrariamente. Podem ser aleatórios.

essa foi a minha resolução:

(COLOQUEI MEU CÓDIGO AQUI)

refaça a minha atividade como vc achar melhor, que nem da ultima vez

**chat:** Boa! Você fez um trabalho bem completo: implementou todos os 8 axiomas da estrutura de espaço vetorial usando `numpy`.

O que achei muito bom:

- Cada axioma tem seu **bloco separado**, com comentário explicando.
- Você gerou vetores aleatórios, então sempre há exemplos diferentes.
- Você comparou lado esquerdo e direito e mostrou se eram iguais.

Mas dá pra melhorar a clareza e evitar repetições.

Por exemplo:

- Muitos `for` são redundantes, já que o `numpy` permite verificar **vetores inteiros de uma vez**.
  - Podemos usar uma **função auxiliar** para imprimir resultados de forma mais limpa.
  - Em vez de repetir `lado_e`, `lado_d` em cada bloco, podemos sempre comparar vetorialmente (`np.array_equal`).
  - Isso deixa o código mais enxuto, organizado e ainda mais didático.
- 

## ✓ Versão refeita e melhorada

Aqui vai uma versão mais clara e enxuta da sua atividade:

```
import numpy as np
```

```
# Função auxiliar para mostrar resultados
```

```
def verificar(nome, lhs, rhs):
```

```
    iguais = np.array_equal(lhs, rhs)
```

```
    print(f"{nome}:")
```

```
    print(f"Lado esquerdo = {lhs}")
```

```
    print(f"Lado direito  = {rhs}")
```

```
    print(f"Iguais? {iguais}\n")
```

```
# 1. Associatividade da adição
```

```
u, v, w = np.random.randint(0, 10, (3, 3))
```

```
verificar("Associatividade da adição  $(u + (v + w) = (u + v) + w)$ ",
```

```
          u + (v + w), (u + v) + w)
```

```
# 2. Comutatividade da adição
```

```

u, v = np.random.randint(0, 10, (2, 3))

verificar("Comutatividade da adição ( $u + v = v + u$ )", u + v, v + u)


# 3. Elemento identidade da adição

v = np.random.randint(0, 10, 3)

verificar("Elemento identidade da adição ( $v + 0 = v$ )", v + 0, v)


# 4. Elemento inverso da adição

v = np.random.randint(0, 10, 3)

verificar("Elemento inverso da adição ( $v + (-v) = 0$ )", v + (-v), np.zeros(3,
dtype=int))


# 5. Compatibilidade da multiplicação por escalar

a, b, v = np.random.randint(1, 10, (3, 3))

verificar("Compatibilidade da multiplicação por escalar ( $a(bv) = (ab)v$ )",

        a * (b * v), (a * b) * v)


# 6. Elemento identidade da multiplicação por escalar

v = np.random.randint(1, 10, 3)

verificar("Elemento identidade da multiplicação por escalar ( $1v = v$ )",

        1 * v, v)


# 7. Distributividade escalar em relação à adição de vetores

```

```

a, u, v = np.random.randint(1, 10, (3, 3))

verificar("Distributividade escalar em relação à adição de vetores  $(a(u+v) = au + av)$ ",

          a * (u + v), a * u + a * v)

# 8. Distributividade escalar em relação à adição do corpo

a, b, v = np.random.randint(1, 10, (3, 3))

verificar("Distributividade escalar em relação à adição do corpo  $((a+b)v = av + bv)$ ",

          (a + b) * v, a * v + b * v)

```

---

## Principais melhorias

1. Criei a função `verificar()` para **padronizar a saída** e evitar repetições.
2. Usei `np.array_equal()` em vez de comparar elemento a elemento.
3. Reduzi a quantidade de variáveis auxiliares, deixando cada bloco mais direto.
4. A saída agora mostra **vetores inteiros de uma vez**, o que facilita a leitura.

### ele está correto?

- Novamente, acho que o chat está correto, mas dessa vez, muito mais eficiente do que o meu código. A resolução do problema proposto foi feita no meu, mas o código saiu repetitivo, ele criou uma função. Usando o `np.array_equal` para facilitar a comparação entre as equações.

