MAC122 – Princípios de Desenvolvimento de Algoritmos
Segundo Semestre de 2025 – BMAC – IMEUSP – Prof. Marcilio
EXERCÍCIO PROGRAMA I – Entregar até 14/setembro/2025

Interpretador de Expressões com Números Complexos

O programa deve ler expressões contendo números complexos na forma **`(a + bi)`** (**`a`** e **`b`** **`int`** ou **`float`**). **`a`** é a parte real e **`b`** a parte imaginária.
As expressões seguem as regras normais de expressões aritméticas com os números complexos na forma **`(a + bi)`**, os operadores binários (+, -, *, /) e os operadores unários (+, -) com a prioridade usual e os parêntesis para alterar a prioridade se necessário.

Deve funcionar como o interpretador Python no modo prompt, mas fazendo as operações com números complexos. Vamos usar o mesmo prompt do Python (>>>).

Exemplo de entrada e saída do programa:

```
>>> (1 + 2i) * (3 + 5i)
(-7 + 11i)
>>>(12 - 5i) - (-2.2 + 3i)/(12 - 1.2i)
(12.20627 - 5.22937i)
>>>(1+0.2i) * ((-3 + 4i) + (4 - 5.2i)) + (2 + 3i)
(3.24 + 1.99999i)
>>>  (-1 + 1.2i) / ((2.3 - 2.3i) + ( 3 + 4i) * (-4 + 5.2i))
(0.02908 - 0.04192i)
>>> -(1+1i) * -(1-1i)
(2 + 0i)
```

Pode supor que a expressão está **sem** erros de sintaxe. Assim não precisa verificar se há caracteres estranhos na expressão.

O programa deve então repetir a sequência abaixo até que seja digitado **`'fim'`**:

- Colocar o prompt **>>>**
- Ler a expressão (string)
- Traduzir a expressão para a notação pós-fixa
- Calcular o valor da expressão
- Mostrar o resultado no vídeo

A tradução para a notação pós-fixa

Use o algoritmo de tradução para pós-fixa que utiliza uma pilha de operadores e os movimenta na expressão baseado em sua prioridade.

Para facilitar a manipulação da string antes de transformá-la em pós fixa, use expressões regulares para separar os elementos da expressão. O método **`findall`** encontra todas as substrings que correspondem a expressão regular, e as retorna como uma lista. Veja o exemplo abaixo

```
import re
# Exemplo 1:
t = " (1 + 2i) * (3 + 5i)"
```

```
r = re.findall(r"(\b\d*[\.]?\d+[i]?\b|[\(\)\+\*\-\/\%])", t)
print(t)
print(r)
print()

# Exemplo 2:
t = "(12 - 5i) - (-2.2 + 3i)/(12 - 1.2i)"
r = re.findall(r"(\b\d*[\.]?\d+[i]?\b|[\(\)\+\*\-\/\%])", t)
print(t)
print(r)
print()

# Exemplo 3:
t = "(1+0.2i) * ((-3 + 4i) + (4 - 5.2i)) + (2 + 3i)"
r = re.findall(r"(\b\d*[\.]?\d+[i]?\b|[\(\)\+\*\-\/\%])", t)
print(t)
print(r)
print()

# Exemplo 4:
t = " -(1+1i) * -(1-1i)"
r = re.findall(r"(\b\d*[\.]?\d+[i]?\b|[\(\)\+\*\-\/\%])", t)
print(t)
print(r)
```

Será impresso:

```
(1 + 2i) * (3 + 5i)
['(', '1', '+', '2i', ')', '*', '(', '3', '+', '5i', ')']

(12 - 5i) - (-2.2 + 3i)/(12 - 1.2i)
['(', '12', '-', '5i', ')', '-', '(', '-', '2.2', '+', '3i', ')', '/',
'(', '12', '-', '1.2i', ')']

(1+0.2i) * ((-3 + 4i) + (4 - 5.2i)) + (2 + 3i)
['(', '1', '+', '0.2i', ')', '*', '(', '(', '-', '3', '+', '4i', ')',
'+', '(', '4', '-', '5.2i', ')', ')', '+', '(', '2', '+', '3i', ')']

 -(1+1i) * -(1-1i)
['-', '(', '1', '+', '1i', ')', '*', '-', '(', '1', '-', '1i', ')']
```

Os operadores e os operandos

Para a tradução, considere a prioridade usual dos operadores:

- + e – unários; / e *; depois + e – binários.
- Parêntesis alteram a prioridade.
- Operadores de mesma prioridade, devem ser tratados da esquerda para a direita.

Os operandos são os números complexos que podem ser armazenados com uma dupla (tupla), em uma lista de 2 elementos ou de qualquer outra forma equivalente. É conveniente já armazená-los como **`float`**.

O cálculo do valor da expressão

Use o algoritmo de cálculo do valor de uma expressão já em notação pós-fixa, que varre a lista com a expressão e usa uma pilha de operandos. Ao final do cálculo (o resultado está no topo da pilha). Os operandos são uma dupla de números.

Como o resultado será sempre **`float`**, na hora de mostrar o resultado, pode-se verificar se o número tem ou não parte fracionária para mostrá-lo como **`int`** ou **`float`**.

Organização do programa

Como sempre, procure estruturar o seu programa de forma modular identificando partes comuns que podem ser reutilizáveis. Faça uma classe **`Pilha`** para implementar tanto a pilha de operadores como a pilha de operandos. A mesma classe **`Pilha`** deve ser usada tanto para traduzir como para calcular o valor da expressão. Faça também uma classe **`Complexo`** que implementa as operações usando a sobrecarga de operadores do Python (+, -, * e / binários e + e – unários).

Faça pelo menos as funções:

a) **`TraduzPosFixa(exp)`** – recebe a string **`exp`** contendo uma expressão aritmética com números complexos e devolve uma lista contendo essa expressão em notação pós-fixa. Esta lista conterá operadores e objetos do tipo **`Complexo`**. Pode considerar que não há erro de sintaxe na expressão, mas se encontrar algum caractere ou algo estranho pode devolver **`None`**.
Exemplo:
Seja **`t = "(2 + 3i) + (5 – 2i) *  -(-3.2 + 5.5i)"`**
`TraduzPosFixa(t)` devolve:
`[(2.0, 3.0), (5.0, -2.0), (-3.2, 5.5), "_", "*", "+"]`
Nesse exemplo considerando os operandos como tuplas. É conveniente substituir o símbolo dos operadores na expressão pós-fixa. Por exemplo: **`"+"`** torna-se **`"#"`** e **`"-"`** torna-se **`"_"`**.

b) **`CalcPosFixa(listaexp)`** – recebe uma lista contendo uma expressão em notação pós-fixa, calcula e devolve o seu valor. Se houver algum problema no cálculo devolve **`None`**.
Exemplo:
Seja `t = [(2.0, 3.0), (5.0, -2.0), (-3.2, 5.5), "_", "*", "+"]`
`CalcPosFixa(t)` devolve o **`Complexo (7.0, -30.9).`**
Imprimir o resultado como:
`(7.0 -30.9i)` ou **`(7 – 30.9i)`**

Como entregar o programa pelo e-disciplinas

Para padronizar a entrega dos programas siga as seguintes diretrizes:

1) O módulo deve se chamar **`posfixa.py`**
2) Deve conter as classes **`Pilha`** e **`Complexo`**.
3) Deve conter também as funções **`TraduzPosFixa`** e **`CalcPosFixa`** como solicitadas acima.
4) Coloque o seu programa principal sob **`if __name__== "__main__":`** para que as funções possam ser testadas com outros programas com **`import`**.
Exemplo:
`if __name__== "__main__":`

```
while True:
    # coloque o prompt
    ...
    # ler a expressão
    ...
    # chamar a TraduzaPosFixa
    ...
    # chamar a CalcPosFixa
    ...
```