



Unidade 7: Utilização da biblioteca Matemática Complexa


Lição 2: Cálculos e representações

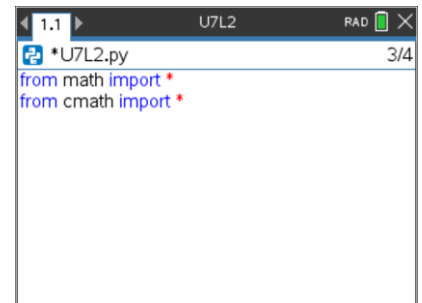
Nesta segunda lição da Unidade 7, aprenderá a utilizar a biblioteca **cmath (Matemática complexa)** para efetuar cálculos simples com números complexos.

Objetivos:

- Utilizar a biblioteca **cmath**.
- Realizar cálculos com números complexos.
- Representar graficamente números complexos

1. Alguns cálculos simples.

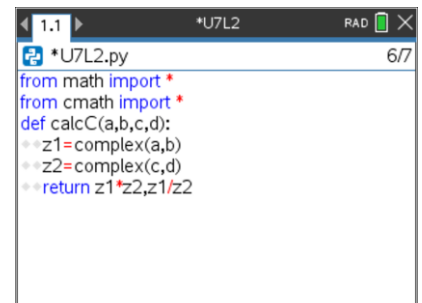
- Iniciar um novo programa com o nome U7L2.
- Inserir uma nova aplicação, escolhendo no menu **A: Adicionar Python**.
- Com a tecla  aceder a **9: Mais módulos** e depois **1: Matemática complexa**.
- Importar também a biblioteca de funções matemáticas.



```

1.1 U7L2 RAD
*U7L2.py 3/4
from math import *
from cmath import *
    
```

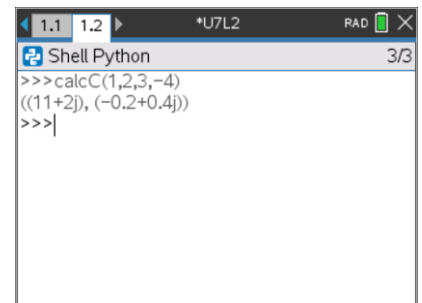
- Criar uma função **calcC(a,b,c,d)** tendo como argumentos as partes reais e imaginárias dos números complexos: $z1 = a + bj$ e $z2 = c + dj$
- Usar esta função para obter o produto e o quociente dos dois números complexos, ou seja, $z1 \times z2$ e $\frac{z1}{z2}$.



```

1.1 U7L2 RAD
*U7L2.py 6/7
from math import *
from cmath import *
def calcC(a,b,c,d):
    z1=complex(a,b)
    z2=complex(c,d)
    return z1*z2,z1/z2
    
```

- Testar a função com dois números complexos à sua escolha.



```

1.1 1.2 U7L2 RAD
Shell Python 3/3
>>> calcC(1,2,3,-4)
((11+2j), (-0.2+0.4j))
>>>
    
```





2. As diferentes formas de um número complexo.

Criará agora uma função que permita trabalhar de forma mais simples com números complexos usando formas trigonométricas ou exponenciais.

a) Forma trigonométrica e exponencial.

Escrever uma função para obter o módulo e o argumento de um número complexo (radianos e graus) para poder escrever na forma

$$z = \rho \times (\cos\theta + j\sin\theta) \text{ e depois } z = \rho \times e^{j\theta}.$$

```

1.1 1.2 *U7L2 RAD 13/13
*U7L2.py
def calcC(a,b,c,d):
    z1=complex(a,b)
    z2=complex(c,d)
    return z1*z2,z1/z2
# Forma trigonométrica
def trigo(a,b):
    z=complex(a,b)
    zz=polar(z)
    módulo=zz[0]
    argumento=degrees(zz[1])
    return round(módulo,3),argumento
    
```

SUGESTÃO:

O módulo e o argumento de um número complexo também se podem determinar utilizando as instruções **abs()** e **phase()** da biblioteca **cmath**.

Estudo de um exemplo:

Considere o número complexo $z = 4\sqrt{3} + 4j$

Determinar o módulo e um argumento deste número complexo (mod 2π).

Dar a expressão na forma trigonométrica e depois exponencial.

A função permite rapidamente verificar que o número complexo tem módulo $\rho = 8$ e argumento $\theta = 30^\circ$, ou $\frac{\pi}{6}$ mod 2π .

A forma exponencial será $z = 8 \times e^{j\frac{\pi}{6}}$.

```

1.1 1.2 *U7L2 RAD 3/3
Shell Python
>>>trigo(4*sqrt(3),4)
(8.0, 30.0)
>>>|
    
```

b) Interesse das formas trigonométricas e exponenciais.

Use as duas funções anteriores (ou crie outra que utilize as duas), para verificar que para dois números complexos:

- O módulo do produto é o produto dos módulos e o argumento do produto é a soma dos argumentos.
- O módulo do quociente é o quociente dos módulos e o argumento do quociente é a diferença dos argumentos.

Pode-se também trabalhar diretamente no interpretador (Shell).

Estudo de um exemplo:

Considere os dois seguintes números complexo na forma algébrica:

$$z1 = 1 + 1j \text{ e } z2 = 4\sqrt{3} + 4j$$

De seguida use os operadores lógicos, **mas tenha cuidado!**

```

1.1 1.2 *U7L2 RAD 11/11
Shell Python
>>>z1=(1+1j)
>>>z2=(4*sqrt(3)+4j)
>>>z1pol=polar(z1)
>>>z2pol=polar(z2)
>>>zmult=z1*z2
>>>zmult_pol=polar(zmult)
>>>zmult_pol[0]
11.31370849898476
>>>z1pol[0]*z2pol[0]
11.31370849898476
>>>
    
```

```

1.1 1.2 *U7L2 RAD 3/3
Shell Python
>>>z1pol[0]*z2pol[0]=zmult_pol[0]
False
>>>|
    
```





3. Representar graficamente um número complexo.

Represente num plano os números complexos anteriores:

$$z1 = 1 + 1j \text{ e } z2 = 4\sqrt{3} + 4j$$

Para tal, deve:

- Extrair as partes reais e imaginárias dos números complexos.
- Guardá-los em duas listas $x[]$ e $y[]$.
- Representar graficamente estas listas como nuvem de pontos.

Inserir um novo programa com o nome U7L21.

Criar uma função para representar graficamente dois números complexos. Esta função pode parecer artificial para representar os afijos de dois complexos z . No entanto, é um primeiro passo para a lição seguinte (Lição 3), na qual poderá resolver uma equação complexa.

```
*U7L21.py 11/18
from cmath import *
from math import *
import tiplotlib as plt
def comp(a,b,c,d):
    z1=complex(a,b)
    z2=complex(c,d)
    x=[z1.real,z2.real]
    y=[z1.imag,z2.imag]
    plt.cls
    plt.window(-10,10,-10,10)
    plt.grid(1,1,"dashed")
```

```
*U7L21.py 19/19
plt.cls
plt.window(-10,10,-10,10)
plt.grid(1,1,"dashed")
plt.axes("on")
plt.labels("parte real","parte imaginária",12,2)
plt.title("Representação de complexos")
plt.color(255,0,0)
plt.scatter(x,y,"o")
plt.show_plot()
```

- Executar o programa.
- Solicitar a representação gráfica dos números complexos propostos.

```
Shell Python 3/3
>>>#Running U7SB21.py
>>>from U7SB21 import *
>>>comp(1,1,4*sqrt(3),4)
```

- Se desejar, pode modificar a representação gráfica para evidenciar o módulo e o argumento (importar eventualmente da biblioteca **TI PlotLib**).

