Politécnico de Leiria

Escola Superior de Tecnologia e Gestão

Matemática Discreta - Componente PL

EI (D)

Ano letivo 2023/2024 - 2.º Sem. Ficha prática 1 (PL)

A partir do ano letivo 2021/2022, a componente laboratorial da unidade curricular (UC) de Matemática Discreta explora os conceitos desta UC recorrendo ao Python, uma linguagem de programação de alto nível cuja utilização tem vindo a ser cada vez mais generalizada em diversas áreas, com notoriedade crescente nos últimos anos.

A aprendizagem de Python não é, por si só, um dos objetivos desta UC; no entanto, sendo esta a linguagem escolhida para implementação dos exercícios da componente laboratorial, torna-se necessário, nesta primeira ficha prática, fazer uma introdução e abordar alguns tópicos essenciais. Ao longo das restantes fichas serão introduzidas instruções adicionais que permitirão resolver os respetivos exercícios.

Introdução ao Python

O Python é uma linguagem de programação de fácil introdução pela simplicidade na sintaxe, para além de ter associada uma larga documentação de suporte à programação. Entre as mais-valias da utilização do Python, destacam-se as seguintes:

* linguagem de alto nível
* linguagem de sintaxe intuitiva que permite o desenvolvimento de projetos e implementação de aplicações complexas, bastante versátil
* linguagem gratuita, de fonte aberta, que corre em diferentes plataformas
* grande quantidade de bibliotecas disponíveis
* várias comunidades de suporte

Onde programar em Python? Existem vários editores e IDE (*integrated development environment*) onde é possível fazê-lo: Colab, PyCharm, Spyder, Jupyter, Sublime Text, entre outros.

No âmbito da UC de Matemática Discreta, o IDE usado será o **Spyder**. Este pode ser instalado através do ambiente Anaconda (`https://www.anaconda.com/`), onde se encontram outros recursos para trabalhar com o Python, ou diretamente através do site (`https://www.spyder-ide.org/`). A utilização do Spyder permitirá guardar e reabrir o código-fonte implementado em aula, executá-lo neste ambiente, ter um suporte para depuração e fazer o realce de sintaxe/preenchimento automático; além disso, este IDE já integra muitas bibliotecas, tais como Matplotlib, Scipy, entre outras.

O ambiente de trabalho do Spyder:

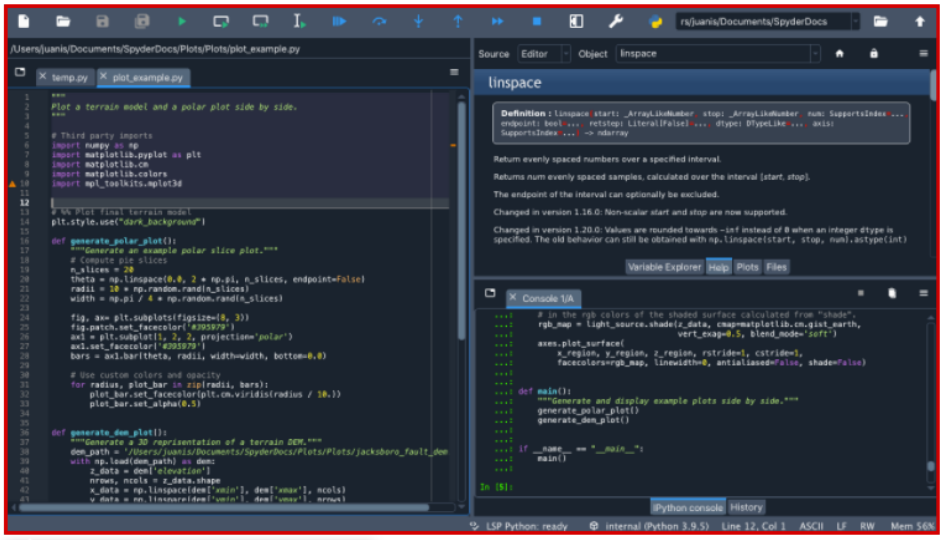
- O **ambiente de trabalho** (padrão) está dividido em vários painéis:

♯ **Editor:** área principal do ambiente de trabalho, onde é escrito/desenvolvido o código em Python. Este editor possui um analisador de código que detecta problemas de estilo, más práticas, entre outros problemas. Ao longo do desenvolvimento do código é possível aceder a opções de preenchimento do mesmo (sugestões/complementos para o que se pretende implementar). As linhas do editor são numeradas e, caso seja detetado um erro de código, este é assinalado na respetiva linha. Se o que se pretende é desenvolver um programa relativamente extenso, será mais eficiente usar o editor, que permite guardar o script desenvolvido, em vez de usar a Consola.

♯ **Consola (Console):** permite executar o código do editor, efetuar cálculos, executar diretamente comandos do Python ou programas armazenados em ficheiros do Python (*.py). Paralelamente a este painel, existe o painel do histórico onde este poderá ser visualizado.

♯ **Outros painéis:**

- **Ajuda (Help):** permite obter documentação relativa aos objetos que são usados no Editor ou na Consola (para a acionar, basta usar o atalho Ctrl+I quando o cursor se encontrar sobre o objeto sobre o qual se pretende pesquisar).
- **Explorador de variáveis (Variable Explorer)**: mostra informação sobre os objetos gerados após a execução do código (por exemplo, variáveis que estão a ser utilizadas). Permite também interagir com estes objetos (por exemplo, redefinindo variáveis).
- **Gráficos (Plots)**: mostra as figuras e gráficos gerados após a execução do código.
- **outras possibilidades**: View -> Panes (escolher opção desejada)



Programação - notas e exemplos iniciais:

- As expressões introduzidas na Janela do Editor são interpretadas e executadas. Em Python, tudo são objetos. Os objetos têm associadas operações que permitem efetuar diversas ações, tais como consultar, construir, modificar, entre outras.

- Para escrever uma linha em **comentário** basta utilizar o símbolo ♯ antes daquilo que se pretende comentar. Assim, quando o código é executado, o que está escrito a seguir ao símbolo ♯, na mesma linha, é ignorado.

  Para comentar múltiplas linhas utilizam-se três aspas (""") (no início e no final do bloco que se pretende comentar).

- Para **executar um comando/linha de comandos** que tenha sido escrito no editor é necessário clicar $Ctrl + return \hookleftarrow$ após a sua definição. Pode ser apenas executada a linha onde se encontra o cursor (ou uma seleção de linhas) pressionando $F9$.

- Os **números** podem ser introduzidos como inteiros (apenas dígitos), como negativos (com um sinal "−" a precedê-los), como decimais (com um ponto a separar a parte inteira da parte fracionária), como complexos (usando j para a unidade imaginária), ou em notação científica (em que as potências de 10 se indicam com a letra e).

- É possível fazer cálculos numéricos simples na consola:

Comandos de cálculo numérico	
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
**	Potênciação

- A **ordem de prioridade das operações** aritméticas é a convencional, ou seja, as potências são executadas antes das divisões e das multiplicações, as quais são realizadas antes das adições e das subtrações. Os parêntesis devem ser usados quando se pretende priorizar operações.

  Exemplos:

```
In [1]: 134*2
Out[1]: 268

In [2]: 3.2*10 / 2**3
Out[2]: 4.0

In [3]: (2+7j)+(8-6j)
Out[3]: (10+1j)

In [4]: 0.00001
Out[4]: 1e-05
```

- Para estabelecer a associação de um nome a um objeto, usa-se o sinal de igual (=). Assim, para atribuir um valor a uma **variável** basta escrever *nome da variável* = *valor da variável*. De forma idêntica, é possível associar um nome a uma expressão mais complexa. É possível também fazer a **atribuição composta**, como ilustram alguns dos seguintes exemplos.

Exemplos:

```
In [5]: x=7; y=4; z=2*x+y;
        z
Out[5]: 18

In [6]: x=x+3  #É equivalente a:
        x+=3

In [7]: y=y*5  #É equivalente a:
        y*=5
```

- Esta linguagem é *case-sensitive,* isto é, faz a diferença entre letras maiúsculas e minúsculas.
- Na tabela seguinte são apresentados comandos úteis na exibição de comentários e na entrada/saída de dados:

Comandos úteis na entrada/saída de dados	
`print`	Exibe comentários, variáveis, ... Quando invocada com mais do que um argumento faz output de todos eles numa só linha, colocando um espaço entre eles.
`str`	Transforma um objeto numa string
+	Concatena strings
`int`	Converte um número ou uma string num número inteiro (se possível)
`float`	Converte um número ou uma string num inteiro decimal (se possível)
`input`	Pede uma entrada de informação através do teclado

Exemplos:

```
In [8]: x=1; print(x)
        1

In [9]: y=3; print("O valor da variável y é",y)
        O valor da variável y é 3

In [10]: y=3; print("O valor da variável y é "+str(y))
         O valor da variável y é 3

In [11]: x="2"; y="7.3"; z=int(x)+float(y);
         print(z)
         9.3

In [12]: x=input("Introduza o valor de x:")
         Introduza o valor de x:
```

Na Consola, o utilizador deve introduzir o valor que deseja atribuir a x. De notar que, no último exemplo, x ficará guardado em formato string.

- A linguagem Python, à semelhança de outras linguagens de programação, contempla extensões à linguagem básica, que permitem realizar as mais variadas operações com objetivos bastantes diversos.
  Existem várias funções e tipos já embutidos no interpretador do Python (por exemplo, as funções listadas na tabela anterior) e estas estão disponíveis por defeito; exemplo disso é também a função `import`, cuja importância será ilustrada de seguida.
  Muitas vezes são necessárias funções (entre outros objetos) que não vêm como parte da instalação padrão, e que se organizam em **bibliotecas**, **packages** ou **módulos**. De acordo com o objetivo pretendido (visualização gráfica, cálculo, análise estatística, entre outros), basta identificar qual o módulo adequado e, no início do *script*, importá-lo, para assim aumentar as capacidades da linguagem de base com vários tipos de itens (subpackages, submódulos, classes, funções ou variáveis) que permitirão concretizar o que se pretende. Por exemplo, se o pretendido for implementar código que envolva funções matemáticas tais como o seno, cosseno, arcsin, entre outras, deve-se importar o módulo `math`, que dá acesso a várias funções matemáticas:

```
In [13]: import math
```

  Após importar o módulo, para aceder ao objeto pretendido (seja ele uma função, uma constante ou outro tipo de objeto) basta escrever

$$nome\_do\_módulo.nome\_do\_objeto$$

- O módulo `math` fornece um conjunto de funções bastante usadas em matemática:

Funções matemáticas elementares	
`math.pow(x,p)`	Potência x^p
`math.exp(x)`	Exponencial de base e de `x`
`math.log(x)`	Logaritmo de base e de `x`
`math.log10(x)`	Logaritmo de base 10 de `x`
`math.log2(x)`	Logaritmo de base 2 de `x`
`abs(x)`	Valor Absoluto de `x`
`math.sin(x)`	Seno de `x`
`math.cos(x)`	Cosseno de `x`
`math.tan(x)`	Tangente de `x`
`math.asin(x)`	Arco seno de `x`
`math.acos(x)`	Arco cosseno de `x`
`math.atan(x)`	Arco tangente de `x`
`round(x)`	Arredonda para o inteiro mais próximo de `x`
`round(x,k)`	Arredonda considerando `k` casas decimais
`max(x1,x2,...)`	Devolve o máximo entre os valores `x1,x2,...`

- O módulo `math` fornece ainda algumas constantes relevantes:

Constantes relevantes e outras notações	
`math.pi`	Número pi (valor aproximado), $\pi = 3.1415927$
`math.e`	Número de Neper (valor aproximado), $e = 2.7182818$
`math.inf`	Infinito
`math.nan`	*Not-a-number*: output gerado quando se tenta calcular `math.inf/math.inf,` por exemplo.

  De notar que, por exemplo, o valor de π é um valor aproximado, já que π é um número irracional.

Exemplos:

```
In [14]: import math

         x=math.sin(math.pi/2)
         print(x)

         y=math.log(math.e**5)
         print(y)
```

```
1.0
5.0
```

- Para verificar exatamente o que foi importado para o script, ou seja, o que determinado objecto contempla e permite fazer, basta recorrer ao comando `dir`:

```
In [15]: dir(math)
```

```
Out[15]: ['__doc__',
          '__loader__',
          '__name__',
          '__package__',
          '__spec__',
          'acos',
          'acosh',
          'asin',
          'asinh',
          'atan',
          'atan2',
          'atanh',
          'ceil',
          'comb',
          'copysign',
          'cos',
          'cosh',
          'degrees',
          'dist',
          'e',
```

(o módulo `math` contempla mais itens que não estão listados na figura anterior)

- Para facilitar a interação com o sistema, é usual associar um *alias* a determinado objeto na fase de importação:

```
In [16]: import math as mt
```

A partir do momento que importamos o módulo `math` estabelecendo esta "nova designação", teremos que a usar para aceder a um determinado objeto deste módulo (em vez de `math`):

```
In [17]: mt.pi
```

```
Out[17]: 3.141592653589793
```

- Podemos também facilitar a interação da seguinte forma:

from *módulo* import *nome_do_objeto* as *novo_nome*

```
In [18]: from math import log as ln

In [19]: ln(mt.e)

Out[19]: 1.0
```

- Para impor condições que permitam executar certo tipo de testes (como, por exemplo, comparar objetos) são necessários **operadores relacionais**, cujo resultado será um objeto do tipo **booleano** (True ou False). Para trabalhar com proposições e implementar certo tipo de condições são também necessários os **operadores lógicos**:

Constantes lógicas	
`True`	Valor lógico *Verdadeiro*
`False`	Valor lógico *Falso*

Operadores Relacionais	
==	igual a
! =	diferente de/não igual a
<	menor que
>	maior que
<=	menor ou igual a
>=	maior ou igual a

Operadores Lógicos	
∼ , `not`	negação
&, `and`	e / conjunção
| , or	ou / disjunção

Exemplo:

```
In [20]: ln(1/2)<0

Out[20]: True

In [21]: ln(mt.e)!=1

Out[21]: False

In [22]: mt.tan(mt.pi/4)<=1

Out[22]: True
```

Exercícios propostos

A resolução dos seguintes exercícios deverá ser implementada em **python** e guardada num ficheiro. Este ficheiro permitirá ao estudante ter um registo das tarefas realizadas em aula.

1. Determine o valor das seguintes expressões numéricas:

   (a) $\dfrac{35.6 \times 64 - 7^3}{45 + 5^2}$;

   (b) $\dfrac{5}{7} \times 4 \times 6^2 - \dfrac{3^7}{9^3 - 236}$;

   (c) $\dfrac{3^2 \times \log(76)}{7^3 + 54} + \sqrt[3]{910}$;

   (d) $\cos^2\left(\dfrac{5\pi}{6}\right) \sin\left(\dfrac{7\pi}{8}\right)^2 + \dfrac{\tan\left(\dfrac{\pi}{6}\ln 8\right)}{\sqrt{7}}$.

2. Defina a variável x como $x = 13.5$ e de seguida determine o valor de:

   (a) $x^3 - 2x + 23.5x^2$;

   (b) $\dfrac{\sqrt{14x^3}}{e^{3x}}$;

   (c) $\log\left|x^2 - x^3\right|$.

3. Sendo $a = 15.62$, $b = -7.08$, $c = 62.5$ e $d = 0.5\,(ab - c)$:

   (a) calcule o seguinte valor:

   $$a + \frac{ab}{c}\frac{(a+d)^2}{\sqrt{|ab|}}$$

   (b) usando o comando `print`, exiba os resultados da alínea anterior de modo que a saída seja

   A solução da alínea 3 a) é $\quad -830.77554$

4. Atribua à proposição p o valor lógico Verdadeiro e à proposição q o valor lógico Falso. Determine o valor lógico da proposição $a = p \vee q$.

5. Sendo p uma proposição verdadeira, q uma proposição falsa e w uma proposição verdadeira:

   (a) determine o valor lógico da proposição $a = p \wedge q$;

   (b) determine o valor lógico da proposição $b = (p \vee q) \wedge (\sim q \wedge w)$.

6. Usando operadores relacionais, verifique o valor lógico das seguintes afirmações:

   (a) "O logaritmo neperiano de 16 é um valor positivo."

   (b) "A tangente de 45 graus é maior do que o seno de 90 graus."