Módulo 1 - Laboratório prático – Exercícios

Em cada módulo, os exercícios assinalados com ♣ são importantes.

1. ♣ Criar um tipo estruturado de dados para guardar a sucessão de números naturais de 1 a 100 estimando o tempo que, em média, o seu sistema despende a criar a dita estrutura de dados.

   Realize esta experiência, estimando o desempenho temporal de cada tipo, repetindo a execução do comando 1000 vezes, usando a função timeit* e estimando o tempo total de execução dos 1000 comandos

   a) usando list;
   b) usando tuple;
   c) usando set;
   d) usando dicionary;

   Compare os tempos de cada tipo e procure justificar as diferenças encontradas.

2. Crie as seguintes variáveis: (i) lista, uma lista com 4 valores inteiros; (ii) vetor, um ndarray com 4 valores inteiros; (iii) matriz, um ndarray com 4 linhas e 3 colunas. Em seguida, execute instruções para:
   - multiplicar o valor 5 a cada uma das estruturas e ver o resultado
   - adicionar a lista com ela mesma e ver o resultado
   - adicionar o array com ele mesmo e ver o resultado
   - adicionar a lista com o array e ver o resultado
   - adicionar a matriz com o vetor e ver o resultado

   Que conclusões é que retira dos resultados? Procure saber mais sobre as potencialidades dos arrays numpy.

3. ♣ Codifique e explique o resultado desta secção de código.

```
lista_simples = [0]*5
lista_de_listas = [[0]*5]*5
print(lista_simples)
print(lista_de_listas)
lista_simples[2] = 1
lista_de_listas[2][2]= 1
print(lista_simples)
print(lista_de_listas)
```

* Documentação em (https://docs.python.org/3/library/timeit.html#timeit.Timer.timeit) e tutorial rápido em https://www.geeksforgeeks.org/timeit-python-examples/ .

4. Um "tuple" é uma sequência imutável de elementos ou uma sequência de elementos imutáveis?

Ilustre a sua resposta com exemplos em código python.

5. Qual o problema deste fragmento de código?

```
elem1 = (1, 2)
conj = set(elem1)
elem2 = ([1,2],3)
conj.add(elem2)
```

Teste que tipos de objetos se podem adicionar a um conjunto, tentando adicionar um conjunto, uma lista, um dicionário e um objeto de uma classe definida por si.

6. Criar funções para executar as seguintes operações:

a) Converter coordenadas cartesianas para coordenadas polares. A função recebe o tuplo (x,y) e devolve o tuplo (r,θ).

b) ♣ Devolver os elementos comuns entre dois dados tuplos. Exemplo:

```
t_intersect((-5, -1, 4), (-5, -2, 4, 5)) →  (-5, 4)
```

c) ♣ Remover duplicados de uma dada lista. Exemplo:

```
l1 = [4, 5]
l2 = [3, 4]
remove_dupls(l1, l2) →  [3, 5]
```

d) Devolver uma lista de resultados de divisões de pares (x, y), sendo os pares dados através de uma lista de tuplos. Deve evitar a divisão por zero e só deve dividir caso a divisão seja maior ou igual a 1. Exemplo:

```
list_res_div((82, 40), (23, 4), (12, 3), (15, 3), (1, 2))→ [2.05, 5.0]
```

i. sem usar *comprehension*

ii. usando *comprehension*.

e) Dadas duas listas, devolver uma lista de pares, juntando cada elemento da 1.ª lista com o elemento da 2.ª na mesma posição, ordenada pelo segundo elemento. Exemplo:

```
lst_triplos([1, 5, 2, 3], [20, 56, 10, 22]) →
                                    [(2, 10), (1, 20), (3, 22), (5, 56)]
```

7. Transponha uma matriz numa única linha de código *python*.

8. ♣ Escreva uma função que recebe um conjunto "c" e um inteiro "n" e devolve um *tuple* contendo n elementos aleatórios **retirados** do conjunto.

```
def ret_elem(c, n):
```

Relembre que objetos da classe *Set* devolvem elementos arbitrários mas **não** aleatoriamente.
Para obter elementos "à sorte" com probabilidade uniforme, use a função *random.sample*.
Nota: não se esqueça de retirar os elementos do conjunto.

9. ♣ Crie uma função para calcular a probabilidade de obter uma sequência de k números consecutivos, retirando aleatoriamente e **sem substituição** elementos de um conjunto composto por n números inteiros consecutivos (k $\leq$ n).

```
def prob(k, n):
```

10. ♣ Escreva um programa para:

a. gerar um conjunto de inteiros entre 0 e 9
b. calcular a probabilidade de obter 3 elementos sucessivamente consecutivos invocando a função `prob(k,n)`.
c. execute um ciclo 10000 vezes invocando a função anterior `ret_elem(c, n)`, onde c é um *set* e n = 3, imprimindo para um ficheiro, a cada 100 ciclos, a média de vezes que obteve 3 números consecutivos em ordem.
d. Observe os seus resultados. Que conclusões consegue retirar?

Exercícios extra

A. **Desenvolver** uma função que devolve a **inversa** de uma matriz.

Relembre Álgebra Linear:

- A matriz *inversa* de uma matriz m quadrada é uma matriz m^{-1} que, multiplicada por m, tem por resultado a matriz identidade da mesma ordem, ou seja, $m\, m^{-1} = I$.
- Método para o desenvolvimento da sua função: **Aplicação da eliminação de Gauss-Jordan**

  (Se desejar, pode ver detalhes em https://pt.wikipedia.org/wiki/Matriz_inversa mas os seus apontamentos de álgebra linear devem funcionar melhor)

B. Usando a função desenvolvida em (**A**), criar uma função

```
resolver_sistemas_equacoes (A, B)
```

em que a matriz dos coeficientes das incógnitas x é a matriz de input A e o vetor dos termos independentes é o vetor de input B.

Assuma que $Ax = B$, logo, $x = B\ A^{-1}$, pelo que basta calcular a inversa de A e fazer a multiplicação matricial desta com B, devolvendo como resultado o vetor resultante do produto.

Verifique a correção do seu código usando `linalg.inv`(A) do módulo `numpy` para a mesma matriz A.