

6.1 Escreva uma função `is_contained(a,b)` que verifica se todos os elementos da lista a estão contidos na lista b , devolvendo `True` nesse caso e `False` no caso contrário. Por exemplo:

```
>>> is_contained([3,4,1],[1,2,4,1,3])
True
```

Comece a sua definição por escrever mais alguns casos de teste.

6.2 Escreva uma função `remadj(xs)` para remover de uma lista elementos adjacentes repetidos, isto é, tais que `xs[i] == xs[i+1]`. O resultado da função deve ser uma nova lista; a lista original não deve ser modificada. Exemplos:

```
>>> remadj([2,2,3,1,4,4])
[2, 3, 1, 4]
>>> remadj(['a','b','b','b','a'])
['a', 'b', 'a']
```

Note que, ao contrário dos exemplos apresentados nas aulas, esta função não deve eliminar *todos* os repetidos. Comece a sua definição por escrever mais alguns casos de teste.

6.3 Escreva uma função `soma_linhas(A,x)` que, dada uma matriz A de $n \times n$ de números inteiros (representada como lista de listas), verifica se a soma dos valores por linha é igual a x . O resultado deverá ser `True` no caso de todas as linhas da matriz terem soma igual a x e `False`, caso contrário.

6.4 Escreva uma função `soma_colunas(A,x)` que, dada uma matriz A de $n \times n$ de números inteiros (representada como lista de listas), verifica se a soma dos valores por coluna é igual a x . O resultado deverá ser `True` no caso de todas as colunas da matriz terem soma igual a x e `False`, caso contrário.

6.5 Um *quadrado mágico* é uma matriz de $n \times n$ que:

1. é preenchida com números inteiros distintos de 1 até n^2 ;
2. todas as linhas, colunas e diagonais somam o mesmo valor.

O exemplo seguinte representa um quadrado mágico em que cada linha, coluna e diagonal soma 15:

2	7	6
9	5	1
4	3	8

Num *quadrado pseudo-mágico* apenas a segunda condição é necessária.

Escreva uma função `pseudo_magico(A)` que testa se uma matriz (representada como lista de listas) é um *quadrado pseudo-mágico*; o resultado deve ser um valor lógico.

6.6 Escreva uma função

```
transposta(A: List[List[int]]) -> List[List[int]]
```

que transforme uma matriz A de dimensões $n \times m$ na sua *transposta* A^T com dimensões $m \times n$, i.e., $A^T_{ij} = A_{ji}$ para todos i, j .

6.7 Uma matriz quadrada A está em forma triangular superior se todos os coeficientes abaixo da diagonal são zero, isto é, se $A_{ij} = 0$ para todos i, j tais que $j < i$. Contudo, por causa da possibilidade de erros de arredondamento, vamos considerar como *zero* os números inferiores em valor absoluto a uma “tolerância” ϵ ; a condição será então $|A_{ij}| < \epsilon$ para todos i, j tais que $j < i$.

Escreva a definição duma função

```
triang_sup(A: List[List[float]], eps: float) -> bool
```

que verifica a condição acima. O resultado deve ser um valor lógico (True/False).

6.8 Escreva uma função `combina(xs, ys)` que, dadas duas listas `xs` e `ys`, combina os elementos das duas listas, dois a dois, numa lista de tuplos. Cada tuplo é formado por um elemento da lista `xs` e por um elemento da lista `ys`. A ordem dos elementos deverá ser a mesma que é apresentada em ambas as listas. No caso das duas listas não terem o mesmo número de elementos, os elementos da lista com menor comprimento deverão ser reciclados. Exemplos:

```
>>> combina(['a', 'b', 'c'], [1, 2, 3, 4, 5])
[('a', 1), ('b', 2), ('c', 3), ('a', 4), ('b', 5)]
```

```
>>> combina(['a', 'b', 'c', 'd', 'e'], [1, 2, 3])
[('a', 1), ('b', 2), ('c', 3), ('d', 1), ('e', 2)]
```

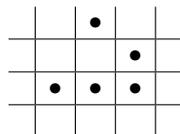
Nota: pode assumir que cada uma das listas `xs` e `ys` têm comprimento maior ou igual a um.

6.9 O jogo *Life* foi inventado pelo matemático britânico John H. Conway e é um dos mais conhecidos exemplos de um autómato celular. Considere um tabuleiro bi-dimensional com linhas horizontais e verticais definindo quadrículas; cada quadrícula pode estar vazia ou conter uma célula. Consideram-se *vizinhos* as oito posições directamente adjacentes a uma quadrícula.

Dada uma configuração inicial, o jogo desenrola-se em gerações sucessivas; as células morrem ou nascem conforme o número de vizinhos:

1. uma célula com menos de dois vizinhos morre de isolamento;
2. uma célula com mais de três vizinhos morre de sobrepopulação;
3. uma célula com dois ou três vizinho continua viva;
4. numa quadrícula vazia com exactamente três vizinhos nasce uma nova célula.

Pretende-se escrever um programa para mostrar a evolução das gerações. Vamos representar uma célula pelo carater '0' e uma quadrícula vazia por '.'; o tabuleiro pode então ser representado por uma lista de cadeias de caracteres. Por exemplo, a seguinte configuração de tabuleiro



será representada por [".0..", "...0.", ".000.", "....."].

Para obter mais informação sobre o jogo *Life* pode consultar a página da Wikipédia: http://en.wikipedia.org/wiki/Conway's_Game_of_Life

6.10 Verifique a resposta do interpretador de *Python* (numa sessão interativa) a cada uma das linhas seguintes:

```
>>> d = {'apples': 15, 'bananas': 35, 'grapes': 12}
>>> d['bananas']
>>> d['oranges'] = 20
>>> len(d)
>>> 'grapes' in d
>>> d['pears']
>>> d.get('pears', 0)
>>> sorted(d)
>>> del d['apples']
>>> 'apples' in d
```

Assegure-se de que percebe cada um dos resultados.

6.11 Escreva uma função `conta_letras(txt)` que imprime uma tabela com o número de ocorrências de cada letra na cadeia de caracteres `txt`, por ordem alfabética. Letras maiúsculas, minúsculas e acentuadas devem ser consideradas iguais. Exemplo:

```
>>> conta_letras("A luz do sol é amarela")
a : 4
d : 1
e : 2
...
```

Nota: Para remover os acentos das letras acentuadas use a função `normalize` do módulo `unicode`.

6.12 Duas palavras ou frases são *anagramas* se, se escrevem com as mesmas letras, usadas o mesmo número de vezes mas, eventualmente, em posições diferentes. Por exemplo, a frase em Latim “*Quid est veritas?*” (*O que é a verdade?*) é um anagrama de “*Est vir qui adest*” (*É o homem que está diante de si*).

Escreva uma função `anagramas(txt1,txt2)` que verifique se as cadeias de caracteres `txt1` e `txt2` são anagramas; o resultado deve ser `True` ou `False`. Deve considerar equivalentes as letras maiúsculas e minúsculas e ignorar todos os caracteres que não são letras (espaços, sinais de pontuação, etc.); pode ainda assumir que as cadeias não têm letras com acentos.

6.13 Escreva uma função `maisFreq(txt)` que, dada uma cadeia de caracteres `txt`, retorna o caracter que ocorre mais vezes em `txt` escrito em *maiúscula*. Para esta contabilização, não deverá ser feita distinção entre letras maiúsculas e minúsculas. No caso de haver mais do que um caracter com o maior valor de ocorrência em `txt`, deve ser retornado o caracter com *menor* ordem lexicográfica.

Exemplos:

