Extra3: Exercícios de treino

Este enunciado tem exercícios semelhantes aos do exame final de FP em 2018-2019. Alguns conselhos:

- É importante **testar** os programas! Deve testar à medida que vai completando cada tarefa.
- Leia e interprete quaisquer erros reportados pelo Python antes de tentar corrigir o programa.
- Enquanto estiver a testar o programa, pode incluir instruções `print` para ver resultados parciais e confirmar a correção da solução. No fim, comente ou apague essas instruções auxiliares.
- Não complique! A maioria dos exercícios tem soluções simples.
- Compare a sua solução com as de outros colegas e analise vantagens e desvantagens.

1. Programa `stocks.py` [6 pontos]

O programa `stocks.py` define uma lista de tuplos com informação sobre acções de diversas empresas transacionadas em bolsas de várias cidades. Cada tuplo contém os campos: empresa, cidade, preço-de-abertura, preço-de-fecho, volume. O programa inclui ainda uma série de instruções para testar a resolução de cada alínea.

a) Altere a função `printStocks` para mostrar a tabela com as colunas alinhadas e formatadas como no exemplo abaixo. Mostre também uma coluna extra com a valorização da ação em percentagem. Por exemplo, se o preço de abertura for 10.00 e o de fecho for 9.50, a valorização será de -5%. Note: esta função não deve modificar a lista dada.

```
INTC      London          34.25    34.45   1792860    0.6%
TSLA      London         221.33   229.63    398520    3.8%
EA        Paris           72.63    68.98   1189510   -5.0%
INTC      Tokyo           33.22    34.29   4509110    3.2%
TSLA      Paris          217.35   217.75    252500    0.2%
ATML      Frankfurt        8.23     8.36    810440    1.6%
```

b) Acrescente os argumentos adequados à função `sorted` para obter uma tabela ordenada alfabeticamente pelo nome da empresa e, para a mesma empresa, por ordem decrescente do volume transacionado.

c) Altere o programa para colocar em `stocks3` uma lista apenas com as ações da bolsa de Paris. Sugestão: pode usar uma *list comprehension*.

d) O ficheiro `stocks.txt` contém informação de mais ações. Cada linha corresponde a uma ação, com os campos separados por TABs. Crie uma função `load` para ler ficheiros com esse formato e devolver uma lista de tuplos do mesmo tipo da variável `stocks`. As instruções após a chamada à função `load` deverão executar sem erros.

2. Programa `arranjos.py` [4 pontos]

O número de arranjos sem reposição de n objetos k-a-k, onde $k \leq n$, pode obter-se pela relação de recorrência abaixo.

$$A(n,k) = \begin{cases} 1 & \text{se } k = 0 \\ n \cdot A(n-1, k-1) & \text{se } 0 < k \leq n \end{cases} . \tag{1}$$

Implemente esta função no programa `arranjos.py` e teste-a, invocando-a para alguns valores de n e k. Por exemplo: $A(2,1) = 2$, $A(5,2) = 20$, $A(5,3) = 60$ e $A(10,3) = 720$.

3. Programa `trains.py` [12 pontos]

O programa `trains.py` permite gerir comboios de mercadorias. Cada comboio é representado por uma lista de vagões e cada vagão é uma lista com dois elementos que indicam o tipo e a quantidade de mercadoria que transporta. Por exemplo,

```
t = [['coal', 30], ['rice', 50], ['iron', 5], ['rice', 42], ['coal', 45]]
```

representa um comboio com 5 vagões: o primeiro vagão com 30 toneladas de carvão, o segundo com 50 toneladas de arroz, etc. O programa já tem uma função `main`, que cria vários comboios e invoca várias funções. Complete as funções para que o programa funcione devidamente.

a) A função `quantityOf(t, m)` deve devolver a quantidade total de mercadoria do tipo `m` contida no comboio `t`.

b) A função `unload(t, m, q)` deve descarregar do comboio `t` uma quantidade `q` de mercadoria de tipo `m`. Para isso, deve percorrer os vagões um a um, a partir do último, e descarregar total ou parcialmente os que tiverem a mercadoria pretendida até perfazer a quantidade pedida. Os vagões totalmente descarregados devem ser retirados do comboio, mas os restantes têm de ficar no comboio pela ordem original. Se conseguir descarregar toda a quantidade pedida, a função deve terminar e devolver zero. Se não, deve devolver a quantidade que ainda falta descarregar.

c) A função `d = merchandise(t)` deve devolver um dicionário `d` com a quantidade total de cada tipo de mercadoria no comboio `t`. O comboio não deve ser alterado.

d) A função principal define um dicionário `trains` que associa nomes a comboios (listas de vagões). Complete a função `trainsPerMerchandise(trains)` para criar um dicionário que, a cada tipo de mercadoria associe os nomes dos comboios que a transportam, sem repetições.

O resultado deve ser semelhante ao seguinte.

```
t: [['coal', 30], ['rice', 50], ['iron', 5], ['rice', 42], ['coal', 45]]

a)
92 5 75 0

b)
unload(t, 'rice', 40) -> 0
t: [['coal', 30], ['rice', 50], ['iron', 5], ['rice', 2], ['coal', 45]]
unload(t, 'coal', 50) -> 0
t: [['coal', 25], ['rice', 50], ['iron', 5], ['rice', 2]]
unload(t, 'iron', 20) -> 15
t: [['coal', 25], ['rice', 50], ['rice', 2]]

c)
{'rice': 52, 'coal': 25}
t: [['coal', 25], ['rice', 50], ['rice', 2]]

d)
trains: {'T1': [['salt', 46], ['sand', 53], ['sand', 11], ['sand', 4], ['iron', 5]],
'T2': [['rice', 51]], 'T3': [['coal', 53]], 'T4': [['sand', 56], ['rice', 53], ['coal', 25]]}
{'iron': {'T1'}, 'sand': {'T4', 'T1'}, 'rice': {'T2', 'T4'}, 'coal': {'T3', 'T4'}, 'salt': {'T1'}}
```