Aula prática nº 9 – Dicionários

Exercícios

1) Escreva um programa que determine a frequência de ocorrência de todas as letras que ocorrem num ficheiro de texto. O nome do ficheiro deve ser passado como argumento na linha de comando (use `sys.argv`). Descarregue "Os Lusíadas" (documento 3333 do Projeto Gutenberg) e faça a contagem. Ajuste o programa para não distinguir maiúsculas de minúsculas. Finalmente, modifique o programa para mostrar o resultado por ordem alfabética. *Sugestão: use `str.isalpha()` para detetar letras e `str.lower()` para converter para minúsculas*.

```
$ python3 countLetters.py pg3333.txt
a 32088
b 2667
c 7672
d 12846
e 33406
...
```

2) O programa `telefone.py`, fornecido em anexo, simula a lista de contactos de um telemóvel, implementada com um dicionário. O programa apresenta um menu com cinco operações. A operação "Listar contactos" já está implementada. Experimente e analise o programa.
   a) Acrescente a operação de "Adicionar contacto". Deve pedir um número e nome, e acrescentá-los ao dicionário.
   b) Acrescente a operação de "Remover contacto". Deve pedir o número e eliminar o item correspondente. (Use o operador `del` ou o método `pop`.)
   c) Acrescente a operação "Procurar Número". Deve pedir um número e mostrar o nome correspondente, se existir, ou o próprio número, caso contrário. *Sugestão: pode recorrer ao método `get`*. (Isto equivale à alínea 3a da aula 06, mas agora usando um dicionário.)
   d) Complete a função `filterPartName`, que dada uma string, deve devolver um dicionário com os contactos (número: nome) cujos nomes incluam essa string. (Similar à alínea 3b da aula 06.) Use essa função para implementar a operação "Procurar Parte do nome", que deve pedir um nome parcial e listar os contactos que o contêm.

3) Adapte o programa anterior para ser possível associar a morada a um contacto. Sugere-se que altere o dicionário para ter pares (`nome, morada`) como *valores* associados às chaves. Altere a função de listagem para mostrar os dados em 3 colunas com larguras fixas, como se vê abaixo: número ajustado à direita, nome centrado na coluna, morada ajustada à esquerda. Use o método `format` das strings. Faça também as adaptações necessárias nas restantes operações.

```
   Numero :           Nome           : Morada
234370200 :  Universidade de Aveiro  : Santiago, Aveiro
876111333 :      Carlos Martins      : Porto
887555987 :        Marta Maia        : Coimbra
```

4) Crie um programa que permita gerir um campeonato de futebol.
   a) O programa deverá pedir ao utilizador os nomes de 4 equipas.

b) O programa deverá determinar que jogos deverão ser realizados (ver exercício da aula 06) e guardá-los numa lista de pares (equipa1, equipa2).
c) O programa deverá perguntar ao utilizador o resultado de cada jogo (golos de cada equipa) e registar essa informação num dicionário indexado pelo jogo. Por exemplo: `resultado[('FCP', 'SLB')] → (3, 2)`.
d) Para cada equipa, o programa deve manter um registo com o número de vitórias, de empates, de derrotas, total de golos marcados e sofridos, e pontos. Com o resultado de cada jogo, deve atualizar os registos das duas equipas envolvidas. O melhor é manter os registos noutro dicionário indexado pela equipa. Por exemplo: `tabela['SLB'] → [0,0,1, 2,3, 0]`.
e) No final, apresente a tabela classificativa com a seguintes colunas: equipa, vitórias, empates, derrotas, golos marcados, golos sofridos e pontos. *Desafio: consegue ordenar a tabela por ordem decrescente de pontos? Faremos isso noutra aula*.
f) Finalmente, deverá apresentar o campeão; a equipa com mais pontos. Em caso de empate de pontos, deverá ser tomado em consideração:
   - Maior número de pontos nos jogos entre as equipas empatadas.
   - Melhor diferença de golos nos jogos entre as equipas.
   - Maior número de golos em todos os jogos destas equipas.

5) O programa `coins.py` contém um conjunto de funções para gerir carteiras de moedas. Cada carteira (*bag*) é representada por um dicionário que a cada tipo de moeda associa o número dessas moedas na carteira. A lista `COINS` contém os tipos de moedas válidas, por ordem decrescente de valor (em cêntimos).
   a) Complete a função `value(b)` para devolver o montante total na carteira `b`.
   b) Complete a função `transfer1coin(b1, c, b2)` para tentar transferir uma moeda de tipo `c` da carteira `b1` para a `b2`. Se `b1` não tiver moedas do tipo `c`, a função deve devolver `False` e deixar as carteiras sem alterações. Se tiver, deve devolver `True` e atualizar o número de moedas nas duas carteiras.
   c) Complete a função `transfer(b1, a, b2)` para tentar transferir um montante `a` de `b1` para `b2`. Deve fazê-lo à custa de várias transferências de uma moeda de cada vez. Se conseguir, a função deve devolver `True` e alterar as carteiras. Se não, deve devolver `False` e manter as carteiras intactas. *Atenção: este é um problema complexo*.
   d) Altere a função `strbag(bag)` para devolver uma string com uma representação mais "amigável", com as quantidades de moedas por ordem decrescente do tipo de moeda, por exemplo.

6) Cada linha do ficheiro `stocks.csv` tem o formato seguinte:
   `Nome, Data, PreçoAbertura, PreçoMaximo, PreçoMinimo, PreçoFecho, Volume`
   Crie um programa que leia esse ficheiro e que determine:
   a) A empresa mais transacionada (com maior volume total).
   b) O dia e valor em que cada ação atingiu o valor mais elevado.
   c) A empresa com maior valorização diária.
   d) A empresa com maior valorização durante o período a que se refere o ficheiro.
   e) Crie uma função que calcule a valorização de um dado portefólio entre duas datas dadas. O portefólio deve ser um dicionário com o número de ações de cada título, e.g.: {'NFLX': 100, 'CSCO': 80}.