CENTRO DE FORMAÇÃO
DE ALCÂNTARA

INTRODUÇÃO À PROGRAMAÇÃO

UFCD(s) 5118, 5119

GUIA DE LABORATÓRIO 3.4
CONTROLO DA EXECUÇÃO - EXTRA (ßeta)

OBJECTIVOS

- Novos exercícios para aplicar a matéria dada e introduzir alguns aspectos novos.

EXERCÍCIOS DE REVISÃO

1. Que funções são automaticamente definidas pelo Python quando cria um gerador?

2. O que são expressões lista (*list comprehensions*) e expressões geradoras? Dê exemplos. Que outros tipos de expressão existem?

3. Para cada uma das situações em baixo mencionadas, escreva dois fragmentos de código para resolver o problema, um utilizando ciclos e outro utilizando as expressões lista/geradoras/etc. mais adequadas:

NOTA: Escreva apenas o código para alcançar o pretendido. Não é necessário escrever um programa completo, com imports, prints, etc.

a) Dada a seguinte lista de números: `nums = [19, 35, 9, 10, 20, 17, 12, 22, 10]` pretende filtrar para uma lista aqueles que são >= 15		
b) Dada a seguinte string `txt = 'ALBERTO'` pretende obter o inverso desta string. (não pode utilizar `reversed`)		
c) Dada uma determinada string, pretende determinar se todas as vogais minúsculas aparecem.		

4. O que é exibido pelas seguinte instruções (se executadas através de um script):

```
cores = ['vermelho', 'verde', 'vermelho',
         'azul', 'verde', 'vermelho']
d = {}
for cor in cores:
    if cor not in d:
        d[cor] = 0
    d[cor] += 1
print(d)
```

```
nomes = ('Alberto Antunes', 'Armando Alves',
         'Alberto Alves', 'Antonio Almeida',
         'Alberto Alexandrino', 'Arnaldo Afonso')

x = (n.split('Alberto')[-1]
     for n in nomes if n.startswith('Alberto'))

for a in x:
    print(a, end='/')
```

```
palavras = ['mesa', 'carro', 'mesa', 'garfo', 'carro',
            'mesa', 'bola', 'pá', 'garfo']

d = {}
for palavra in palavras:
    d[palavra] = d.get(palavra, 0) + 1

print(*d, sep=',', end='=')
print(*d.values(), sep=',')
```

```
soma = 0
with open('extra.txt') as fich:
    for linha in fich:
        try:
            partes = linha.split()
            soma += float(partes[1])
        except ValueError:
            print("ERRO:", linha, end='')
        else:
            print(partes[0], '->', partes[1])
print(soma)
```

NOTA:

Assuma que o ficheiro `extra.txt` possui:

`Alberto 19`		`Alberto 19`		`Alberto 19`
`Armando 24`	… e depois …	`Armando x24`	… e depois …	`Armando`
`António 150`		`António 150`		`António 150`

5. Considere a lista `nomes = ['Alberto', 'David', 'Armando', 'Raquel', 'Diana', 'Diogo', 'Jose']`. Escreva o código para agrupar estes nomes num dicionário por tamanho. Para a lista `nomes`, o seu algoritmo devolver: `{4: ['Jose'], 5: ['David', 'Diana', 'Diogo'], 6: ['Raquel'], 7: ['Alberto', 'Armando']}`.

EXERCÍCIOS DE PROGRAMAÇÃO

6. A BD de um determinado clube de vídeos consiste de um ficheiro em formato CSV com os seguintes campos por linha: `ID` (numérico), `titulo` (texto), `realizador` (texto), `genero` (texto), `data` e `duração` em minutos. Linhas iniciadas com `#`, `;` ou `//` são consideradas comentários e devem ser ignoradas. Linhas em branco ou apenas com pontos também devem ser ignoradas.

Deverá fazer um programa que lê este catálogo para memória para uma lista de "objectos", cada qual representado com um dicionário. Inicialmente, o programa deve exibir um menu semelhante ao seguinte:

```
CATÁLOGO:
E. Exibir catálogo
I. Exibir linhas inválidas
P. Pesquisar catálogo

>>
```

A opção **E** exibe todo o catálogo em formato legível e a opção **I** exibe apenas as linhas inválidas. A opção **P** deve dar origem ao seguinte menu:

```
PESQUISAR CATÁLOGO
1. Pesquisar por título
2. Pesquisar por género
3. Pesquisar por ano
4. Pesquisar por autor

>>
```

Fica ao seu critério a forma de interacção com o utilizador para efectuar as pesquisas e nas restantes situações não especificadas.

O programa deve também disponibilizar uma interface da linha de comandos semelhante à seguinte:

```
$ bdvideos [exibir | pesquisar (-t titulo | -g genero) | invalidas]
```

Quando invocado desta forma, o programa não exibe menu (ie, não arranca em modo interactivo). O seu programa deverá estar preparado para eventuais situações de erro.

7. Investigue o módulo `zipfile` e desenvolva um *script* que lhe permita comprimir um conjunto de caminhos especificados a partir da linha de comandos. Cada um desses caminhos pode ser para um ficheiro ou para uma pasta (em caso de pasta, deve comprimir os ficheiros aí residentes). O seu *script* deve suportar os algoritmos ZIP, BZIP2 e LZMA (note que se utilizar `zipfile` a extensão será sempre `.zip`).

A interface da linha de comandos deverá ser semelhante a:

```
$ encolhe [(-z | -b | -l)] arquivo caminho1 [caminho2 caminho3 ...]
```

Os parâmetros `-z`, `-b` e `-l` designam os algoritmos de compressão. Quando não indicado na linha de comandos, o programa utiliza o algoritmo por omissão do formato ZIP, ZIP_DEFLATED, e que corresponde ao parâmetro `-z`. O parâmetro `arquivo` indica o nome do arquivo ZIP a criar na pasta actual.

Pode testar a descompressão utilizando as aplicações apropriadas (*unzip*, *7z*, *WinZIP*, *WinRAR*, etc.) ou pode desenvolver o "descompressor".