Algoritmos e Técnicas de Programação
Engenharia Biomédica (2º ano)
Teste (época normal)

5 de Janeiro de 2022 (13h00)

Dispõe de **2:00 horas** para realizar este teste.

Questão 1 (3v = 1+1+1)

Especifique as seguintes listas em compreensão:

a) Lista formada pelos cubos dos números inteiros entre 1 e 20 inclusive:

```
lista = [...]
```

b) Lista formada pelas palavras do texto que contêm a letra 'a':

```
texto = """Vivia há já não poucos anos algures num concelho do Ribatejo
um pequeno lavrador e negociante de gado chamado Manuel Peres Vigário"""
lista = [...]
```

c) Lista formada pelos números inteiros entre 1 e 300 inclusive que são divisíveis por 13:

```
lista = [...]
```

Questão 2 (6v = 1.5+1.5+1.5+1.5)

À semelhança do que foi feito nas aulas, realize as seguintes tarefas:

a) Especifique uma função que calcula e retorna a maior palavra de um texto passado como parâmetro (se o texto for a string vazia a função deverá retornar uma string vazia):

```
def maiorPal(texto):
    ...
    return ...
```

b) Especifique uma função que calcula recursivamente o maior duma lista de inteiros passada como parâmetro (se a lista for vazia o valor None deverá ser retornado):

```
def maior(lista):
    ...
    return ...
```

c) Especifique uma função que calcula o menor múltiplo comum de dois números inteiros passados como parâmetro:

```
def mmc(a, b):
    ...
    return ...
```

d) Especifique uma função que calcula o menor múltiplo comum dos números inteiros pertencentes à lista passada como parâmetro (pode usar a função definida na alínea anterior):

```
def mmcLista(lista):
   ...
   return ...
```

Questão 3 (6v = 0.5+0.5+0.5+0.5+1+1+1+1)

Considere que a informação sobre um stock de uma loja está armazenada numa lista de tuplos (com o nome do produto, o seu preço unitário, e a quantidade em stock desse produto) de acordo com as seguintes estruturas em comentário:

```
# Produto = (nomeProd, preco, quantidade)
# designacao = String
# preco = Float
# quantidade = Int
#
# Stock = [Produto]
#
# Instância exemplo de uma papelaria
meuStock = [("caneta", 2.3, 34), ("lápis", 1.6, 22), ("caderno A4", 3.5, 50)]
```

Assuma que um produto não ocorre mais do que uma vez na lista de stock.

Defina as seguintes funções de manipulação e consulta do stock:

a) `quantos`, que indica quantos produtos existem em stock:

```
def quantos(stock):
    ...
    return ...
```

b) `emStock`, que indica a quantidade de um dado produto existente em stock. Se o produto não existir a função deverá retornar 0:

```
def emStock(nomeProd, stock):
    ...
    return ...
```

c) `consulta`, que indica o preço de um dado produto existente em stock. Se o produto não existir a função deverá retornar 0:

```
def consulta(nomeProd, stock):
    ...
    return ...
```

d) `tabPrecos`, que coloca uma tabela de preços (2 colunas: nome do produto e preço) na consola de output.

```
def tabPrecos(stock):
    ...
    return ...
```

e) `valorTotal`, que calcula o valor total do stock (se vendêssemos o stock inteiro quanto realizaríamos em dinheiro).

```
def valorTotal(stock):
    ...
    return ...
```

f) `inflacao`, que recebe uma taxa de inflação (número decimal entre 0 e 1), e vai atualizar os preços todos multiplicando-os por $1 + taxa$, devolvendo um novo Stock.

```
def inflacao(taxa, stock):
    ...
    return ...
```

g) `maisBaratos`, que indica o(s) produto(s) mais barato(s) (se houver mais do que um produto com o preço mais baixo o resultado será uma lista desses produtos, se houver apenas um produto com o preço mais baixo o resultado será uma lista apenas com esse produto).

```
def maisBaratos(stock):
    ...
    return ...
```

h) `maisCaros`, que constrói a lista dos produtos caros (i.e., acima de um dado preço fornecido como parâmetro).

```
def maisCaros(valor, stock):
    ...
    return ...
```

Questão 4 (5v = 2+2+1)

No seguimento da pergunta anterior, considere o seguinte modelo de uma lista de compras:

```
# ListaCompras = [(nomeProd, quantidade)]
```

E desenvolva as seguintes alíneas:

a) `verifLista`, que verifica se a lista de compras pode ser satisfeita, devolve como resultado 1 se sim e 0 caso contrário. Assuma que na lista de compras não há produtos repetidos.

```
def verifLista(lcompras, stock):
    ...
    return ...
```

b) `falhas`, que devolve a lista de itens (estrutura igual à lista de compras) que não podem ser comprados por não haver Stock suficiente na loja.

```
def falhas(lcompras, stock):
    ...
    return ...
```

c) `custoTotal`, que calcula o custo total da lista de compras.

```
def custoTotal(lcompras, stock):
    ...
    return ...
```