Algoritmos e Técnicas de Programação
Engenharia Biomédica (2º ano)
Teste (época de recurso)
24 de Janeiro de 2023 (14h00)

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 elementos comuns a duas listas:

```
lista1 = [1, 2, 3, 4, 5]
lista2 = [4, 5, 6, 7, 8]
comuns = [...]
# Resultado esperado: [4, 5]
```

b) Lista formada pelas primeira letra de cada palavra do texto:

```
texto = """Vivia há já não poucos anos algures num concelho do Ribatejo
um pequeno lavrador e negociante de gado chamado Manuel Peres Vigário"""
lista = [...]
# Resultado esperado: ['V', 'h', 'j', 'n', ...]
```

c) Lista formada por todas as combinações possíveis de duas letras provenientes de duas listas de letras:

```
letras1 = ['a', 'b', 'c']
letras2 = ['x', 'y', 'z']
combinações = [...]
# Resultado esperado: [('a','x'), ('a','y'), ('a','z'), ('b','x'), ...]
```

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 dada uma string e uma substring não vazia, calcula recursivamente o número de vezes que a substring aparece na string, sem que haja sobreposição de substrings:

```
def strCount(s, subs):
   ...
   return ...

strCount("catcowcat", "cat") --> 2
strCount("catcowcat", "cow") --> 1
strCount("catcowcat", "dog") --> 0
```

b) Especifique uma função que recebe uma lista de números inteiros positivos e devolve o maior produto que for possível calcular multiplicando os 3 maiores inteiros da lista:

```
def produtoM3(lista):
    ...
    return ...

print(produtoM3([12,3,7,10,12,8,9]))
# Resultado esperado: 1440
```

c) Especifique uma função que dado um número inteiro positivo, repetidamente adiciona os seus dígitos até obter apenas um dígito que é retornado como resultado:

```
Input: 38
Output: 2
Explicação: 3 + 8 = 11, 1 + 1 = 2.

Input: 777
Output: 3
Explicação: 7 + 7 + 7 = 21, 2 + 1 = 3.

    def reduxInt(n):
        ...
        return ...
```

d) Especifique uma função que recebe uma string e um número inteiro positivo n e retorna uma string resultante de copiar n vezes a string original:

```
Input: "Hello", 2
Output: "HelloHello"

Input: "abc", 3
Output: "abcabcabc"

    def strNcopy(s, n):
        ...
        return ...
```

Questão 3 (5v = 1 + 1 + 1 + 1 + 1)

Considere que a informação sobre uma agenda de contactos está armazenada numa lista de dicionários (com chaves *nome* e *telefone*) de acordo com as seguintes estruturas em comentário:

```
# Contacto = {'nome': nome, 'telefone': telefone}
# nome = String
# telefone = String
#
# Agenda = [Contacto]
#
# Instância exemplo de uma agenda
minhaAgenda = [{"nome": "Alice", "telefone": "253604455"}, {"nome": "Berto", "telefone": "253604100"}]
```

Defina as seguintes funções de manipulação e consulta da agenda:

a) `quantosLocal`, que indica quantos contactos são de uma dada localidade:

```
def quantosLocal(agenda, prefixo):
    ...
    return ...

# Quantos contactos de Braga
print(quantosLocal(minhaAgenda, "253"))
```

b) `consultaNum`, que devolve o número de telefone associado ao nome passado como parâmetro, se o nome não existir deverá ser devolvida a constante *None*:

```
def consultaNum(agenda, nome):
    ...
    return ...
```

c) `listagem`, que lista a agenda na consola de output ordenada alfabeticamente por nome:

```
def listagem(agenda):
    ...
    return ...
```

d) `insContacto`, que acrescenta um contacto à agenda a partir dos parâmetros recebidos.

```
def insContacto(agenda, nome, telefone):
    ...
    return ...
```

e) `remContacto`, que remove um contacto da agenda, correspondente ao nome recebido.

```
def remContacto(agenda, nome):
    ...
    return ...
```

Questão 4 (6v = 1 + 1 + 1 + 1 + 1 + 1)

Considere que a informação sobre uma lista de tarefas a realizar está armazenada numa lista de dicionários (com chaves *tarefa* e *realizada*) de acordo com as seguintes estruturas em comentário:

```
# Tarefa = {'tarefa': designacao, 'realizada': estado}
# designacao = String
# estado = "sim" ou "não"
#
# ListaTarefas = [Tarefa]
#
# Instância exemplo de uma lista de tarefas
tarefas = [{"tarefa": "Comprar pão", "realizada": "não"}, {"tarefa": "Lavar o carro", "realizada": "sim"}
```

Defina as seguintes funções de manipulação da lista de tarefas:

a) `adicionarTarefa`, que adiciona uma tarefa à lista de tarefas retornando uma nova lista de tarefas. Considere que sempre que uma tarefa nova é adicionada, o campo *realizada* tem valor `"não"`.

```
def adicionarTarefa(ltarefas, designacao):
    ...
    return ...
```

b) `marcarRealizada`, que marca uma tarefa como realizada devolvendo a nova lista de tarefas.

```
def marcarRealizada(ltarefas, designacao):
    ...
    return ...
```

c) `removeTarefa`, que remove uma tarefa da lista.

```
def removeTarefa(ltarefas, designacao):
    ...
    return ...
```

d) `listarRealizadas`, que lista na consola de output as tarefas realizadas.

```
def listarRealizadas(ltarefas):
    ...
    return ...
```

e) `listarTarefas`, que devolve um dicionário com a seguinte estrutura:

```
# Resultados esperado:
#   {
      'realizadas': ["designacao da tarefa x", "designacao da tarefa y", ...],
#     'porRealizar': ["designacao da tarefa a", "designacao da tarefa b", ...]
#   }

def listarTarefas(ltarefas):
    ...
    return ...
```

f) `taxaExecucao`, que devolve um tuplo em que o primeiro elemento é a percentagem de tarefas realizadas e o segundo elemento é a percentagem de tarefas a realizar:

```
# Resultados esperado:
#   (percentagem_tarefas_realizadas, percentagem_tarefas_por_realizar)

def taxaExecucao(ltarefas):
   ...
   return ...
```