Lista_03_Solucoes

September 22, 2020

1 Lista de Exercícios 3 (soluções propostas)

*Alguns exercícios foram retirados do livro: The Python Workbook - Ben Stephenson.

Não é um gabarito fechado. As soluções encontradas aqui são apenas algumas possibilidades. Usem a criatividade!

[1]:
```
import random as rd
```

1.1 QUESTÃO 1:

Crie uma função que retorne uma lista contendo todas as sublistas possíveis dessa lista. Por exemplo, as sublistas de $[1, 2, 3]$ são [], $[1]$, $[2]$, $[3]$, $[1, 2]$, $[2, 3]$ e $[1, 2, 3]$. Observe que sua função sempre retornará uma lista contendo pelo menos a lista vazia porque a lista vazia é uma sublista de todas as listas.

[2]:
```
def todasSublistas(lista):
    sublistas=[[]]
    for i in range(1, len(lista)+1):
        for j in range(len(lista)-i+1):
            sublistas.append(lista[j:j+i])
    return sublistas
```

[3]:
```
todasSublistas([1,2,3])
```

[3]:
```
[[], [1], [2], [3], [1, 2], [2, 3], [1, 2, 3]]
```

1.2 QUESTÃO 2:

Duas palavras são **anagramas** se possuírem todas as mesmas letras (e na mesma quantidade), mas em uma ordem diferente. Por exemplo, “amor” e “roma” são anagramas, assim como “iracema” e “america”. Note que não estamos levando em consideração letras maiúsculas nem acentuação.

Sua tarefa é criar uma função que recebe como parâmetros de entrada duas palavras e que determina se são ou não anagramas.

DICA 1: Talvez seja melhor você criar uma função auxiliar que conta a quatidade de ocorrência de cada letra. Pense em uma função em que a saída seja uma lista de listas, em que cada sublista é formada por dois elementos: o primeiro é a letra e o segundo a quantidade de vezes em que ela aparece na palavra.

DICA 2: Não é necessário, mas se você quiser fazer um tratamento da palavra antes de executar o código, você pode usar o procedimento `.lower()` para tornar todas as letras minúsculas. Também existe a função `unidecode()` da biblioteca `unidecode` que remove o acentos da palavra, mas essa biblioteca pode precisar de instalação. Por hora, não é necessário se preocupar com isso.

[4]:
```
#!pip3 install unidecode            #Caso a biblioteca não esteja instalada,␣
↪execute essa linha!
from unidecode import unidecode     #Importa somente a função unidecode da␣
↪biblioteca unidecode
```

[5]:
```
unidecode('áéíóúàç')
```

[5]:
```
'aeiouac'
```

[6]:
```
def contaLetras(palavra):
    #todasLetras=list(unidecode(palavra.lower())) #Convertemos as letras␣
↪maiúsculas em minúsculas e acentos!
    todasLetras=list(palavra.lower())            #Convertemos apenas as letras␣
↪maiúsculas em minúsculas!
    letras=[]
    contagem=[]
    for letra in todasLetras:
        if letra not in letras:
            letras.append(letra)
            ocorrencias=todasLetras.count(letra)
            contagem.append([letra,ocorrencias])
    contagem.sort()
    return contagem
```

[7]:
```
contaLetras('America')
```

[7]:
```
[['a', 2], ['c', 1], ['e', 1], ['i', 1], ['m', 1], ['r', 1]]
```

[8]:
```
def anagramas(palavra1,palavra2):
    cont1=contaLetras(palavra1)
    cont2=contaLetras(palavra2)
    if palavra1!=palavra2 and cont1==cont2:
        return True
    else:
        return False
```

[9]:
```
anagramas('Iracema','America')
```

[9]:
```
True
```

1.3 QUESTÃO 3:

Crie uma função que simule um lançamento de dados. Lance o dado 100 vezes e armazene os resultados em um lista de listas que mostra a quantidade de vezes cada valor (de 1 a 6) foi obtido. Ou seja, cada sublista é formada por dois elementos: o valor e o número de vezes que ele apareceu.

Em seguida, crie outra função que encontra qual o valor que saiu mais vezes e qual foi essa quantidade (ou seja, a saída é uma tupla ou uma lista de dois elementos).

[10]:
```
def dados():
    #resultados=[[i,0] for i in list(range(1,7))] #Outra forma de construir a␣
 ↪lista de resultados sem usar append!
    resultados=[]
    for i in range(1,7):
        resultados.append([i,0])
    for i in range(100):
        jogada=rd.randint(1,6)
        resultados[jogada-1][1]=resultados[jogada-1][1]+1
    return resultados
```

[11]:
```
resultado=dados()
print(resultado)
```

```
[[1, 13], [2, 23], [3, 11], [4, 16], [5, 20], [6, 17]]
```

[12]:
```
def maiorResultado(resultados):
    valor=resultados[0][0]
    maximo=resultados[0][1]
    for i in range(1,len(resultados)):
        aux=resultados[i][1]
        if aux>maximo:
            valor=resultados[i][0]
            maximo=aux
    return [valor,maximo]
```

[13]:
```
maiorResultado(resultado)
```

[13]:
```
[2, 23]
```

1.4 QUESTÃO 4:

Inicialmente, crie uma função que sorteia n (parâmetro de entrada) cartas de um baralho, podendo ocorrer repetição.

Em seguida cria outra função que sorteia essas n cartas sem que ocorra repetição. Uma **dica**, nesse caso, é criar um baralho como uma lista de listas, ou seja, cada sublista representa uma carta, em que o primeiro elemento é o número da carta e o segundo, o naipe dela.

Um baralho é formado por 52 cartas (números de 2 a 10, Valete (11), Rainha (12), Rei (13) e Ás (1)), dividido em 4 naipes (copas, ouros, paus e espadas).

DICA 1: As cartas nomeadas também correspondem a números. Trabalhe com as cartas variando de 1 a 13.

DICA 2: Na biblioteca `random` existe a função `choice()`, que recebe como parâmetro de entrada uma lista e sorteia aleatoriamente um dos elementos dela. Você pode usar essa função para sortear o naipe de uma carta.

[14]:
```
naipe=rd.choice(['copas','ouros','paus','espadas'])
naipe
```

[14]:
```
'paus'
```

[15]:
```
def baralhoComRepeticao(n):
    #Sorteia n cartas, podendo ocorrer repetição.
    lista=[]
    for i in range(n):
        carta=rd.randint(1,13)
        naipe=rd.choice(['copas','ouros','paus','espadas'])
        lista.append([carta,naipe])
    return lista
```

[16]:
```
baralhoComRepeticao(10)
```

[16]:
```
[[8, 'espadas'],
 [10, 'ouros'],
 [3, 'ouros'],
 [10, 'espadas'],
 [3, 'espadas'],
 [8, 'ouros'],
 [7, 'paus'],
 [8, 'copas'],
 [5, 'paus'],
 [6, 'copas']]
```

[17]:
```
def baralhoSemRepeticao(n):
    #Sorteia n cartas, SEM repetição.
    if n>52:
        print('Não dá pra sortear mais cartas do que existem!')
        return []
    naipes=['copas','ouros','paus','espadas']
    baralho=[]
    lista=[]
    #Cria o baralho com as 52 cartas
    for carta in range(1,14):
        for naipe in range(4):
            x=[carta,naipes[naipe]]
            baralho.append(x)
    #Começa o sorteio das n cartas
```

```
    for i in range (n):
        cartasorteada=baralho.pop(rd.randint(0,len(baralho)-1)) #sorteia o
↪índice da lista e faz o pop
        lista.append(cartasorteada) #insere o elemento retirado do baralho na
↪lista de sorteadas
    return lista
```

[18]:
```
baralhoSemRepeticao(10)
```

[18]:
```
[[8, 'copas'],
 [2, 'copas'],
 [4, 'copas'],
 [3, 'espadas'],
 [8, 'ouros'],
 [13, 'copas'],
 [11, 'espadas'],
 [4, 'espadas'],
 [3, 'copas'],
 [9, 'espadas']]
```

1.5 QUESTÃO 5:

Na lista anterior (questão 6 da lista 2), foi pedido que você criasse uma função que encontrasse o número mínimo de vezes que você tem que jogar uma moeda antes de poder ter três lançamentos consecutivos que resultassem no mesmo resultado (ou todos os três são CARA (C) ou todos os três são COROA (K)).

Agora, crie uma função que execute uma simulação para saber quantos lançamentos são necessários fazer em média para obter tal feito? Para isso, você pode executar a função criada umas 100 vezes e calcular a média dos resultados.

[19]:
```
def moedas():
    ops = ['C', 'K'] #Já podemos usar a ideia de listas!
    resultado = ''
    while True:
        moeda=rd.choice(ops) #Faz um aleatório entre os elementos da lista!
        resultado=resultado+moeda

        if len(resultado) >= 3:
            if ( resultado[-3] == resultado[-2] == resultado[-1] ):
                break
            else:
                continue
    return [resultado, len(resultado)]
```

[20]:
```
moedas()
```

[20]:
```
['KKCKCCC', 7]
```

[21]:
```
def simulaMoedas(rep):
    resultados=[]
    soma=0
    for i in range(rep):
        resultados.append(moedas())
        soma=soma+resultados[i][1]

    print(*resultados, sep="\n") #Para o print sair com cada elemento da lista␣
 ↪em linhas diferentes!

    media = soma/rep
    return media
```

[22]:
```
#simulaMoedas(100)
simulaMoedas(10)
```

```
['KKK', 3]
['CKKCCC', 6]
['KKK', 3]
['CKKCCKKCKCCC', 12]
['CKKCKKK', 7]
['KCCKKCKKCKCCKKCCC', 17]
['KCKKCKKK', 8]
['KKK', 3]
['KKK', 3]
['KCCKKCCC', 8]
```

[22]: 7.0