

Lista_05_Solucoes

October 26, 2020

1 Lista de Exercícios 5 (soluções propostas)

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

```
[1]: import numpy as np
import pandas as pd #biblioteca usada para importar os dados usados na questão 9
```

1.1 QUESTÃO 1:

Implementar a ordenação de matrizes pelo método do [Bubble Sort](#). A função deve receber dois parâmetros de entrada: (i) uma matriz e (ii) o tipo de ordenamento (linha ou coluna).

```
[2]: def bubbleSort(matriz, tipo):
    if tipo != 'linha' and tipo != 'coluna':
        print("Erro: escolha o ordenamento por 'linha' ou 'coluna'!")
        return

    mat = matriz.copy()
    nlin = mat.shape[0]
    ncol = mat.shape[1]

    if tipo == 'linha':
        for i in range(nlin):
            houvetroca = 1
            passadas = 0 #serve para controlar o loop e evitar que vá até o final
            → da lista desnecessariamente
            while houvetroca == 1:
                houvetroca = 0
                for j in range(ncol - 1 - passadas):
                    if mat[i, j] > mat[i, j + 1]:
                        mat[i, j], mat[i, j + 1] = mat[i, j + 1], mat[i, j]
                        houvetroca = 1
                passadas = passadas + 1
    elif tipo == 'coluna':
        for j in range(ncol):
            houvetroca = 1
```

```

        passadas=0 #serve para controlar o loop e evitar que vá até o final
        ↳ da lista desnecessariamente
        while houvetroca==1:
            houvetroca=0
            for i in range(nlin-1-passadas):
                if mat[i,j]>mat[i+1,j]:
                    mat[i,j],mat[i+1,j]=mat[i+1,j],mat[i,j]
                    houvetroca=1
            passadas=passadas+1

    return mat

```

```

[3]: x=np.array([[5,1,10,2,8,4,3,21],[5,0,11,1,7,2,3,20]])
x

```

```

[3]: array([[ 5,  1, 10,  2,  8,  4,  3, 21],
           [ 5,  0, 11,  1,  7,  2,  3, 20]])

```

```

[4]: bubbleSort(x,'f')

```

Erro: escolha o ordenamento por 'linha' ou 'coluna'!

```

[5]: bubbleSort(x,'linha')

```

```

[5]: array([[ 1,  2,  3,  4,  5,  8, 10, 21],
           [ 0,  1,  2,  3,  5,  7, 11, 20]])

```

```

[6]: bubbleSort(x,'coluna')

```

```

[6]: array([[ 5,  0, 10,  1,  7,  2,  3, 20],
           [ 5,  1, 11,  2,  8,  4,  3, 21]])

```

1.2 QUESTÃO 2:

Criar uma função que recebe uma matriz como parâmetro de entrada e troca 2 linhas (ou colunas) escolhidas (parâmetros de entrada) de posição. A escolha da troca por 'linha' ou 'coluna' também é um parâmetro de entrada.

```

[7]: def trocaLinhasOuColunas(matriz,troca,t1,t2):
    if troca!='linha' and troca !='coluna':
        print("Erro! Escolha se a troca é por 'linha' ou 'coluna'!")
        return

    mat=matriz.copy()
    nlin,ncol=mat.shape
    #nlin=mat.shape[0]
    #ncol=mat.shape[1]

```

```

if troca=='linha':
    #verifica se as LINHAS t1 e t2 existem na matriz mat
    if t1>nlin-1 or t2>nlin-1:
        print('Erro! Digite posições de linhas válidas para a matriz_
↪escolhida!')
        return
    #troca as LINHAS t1 e t2 da matriz mat de posição
    for j in range (ncol):
        mat[t1,j],mat[t2,j]=mat[t2,j],mat[t1,j]

elif troca=='coluna':
    #verifica se as COLUNAS t1 e t2 existem na matriz mat
    if t1>ncol-1 or t2>ncol-1:
        print('Erro! Digite posições de colunas válidas para a matriz_
↪escolhida!')
        return
    #troca as COLUNAS t1 e t2 da matriz mat de posição
    for i in range (nlin):
        mat[i,t1],mat[i,t2]=mat[i,t2],mat[i,t1]

return mat

```

```

[8]: tr=np.matrix([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16],[17,18,19,20]])
tr

```

```

[8]: matrix([[ 1,  2,  3,  4],
             [ 5,  6,  7,  8],
             [ 9, 10, 11, 12],
             [13, 14, 15, 16],
             [17, 18, 19, 20]])

```

```

[9]: trocaLinhasOuColunas(tr,'linha',4,0)

```

```

[9]: matrix([[17, 18, 19, 20],
             [ 5,  6,  7,  8],
             [ 9, 10, 11, 12],
             [13, 14, 15, 16],
             [ 1,  2,  3,  4]])

```

```

[10]: trocaLinhasOuColunas(tr,'coluna',1,3)

```

```

[10]: matrix([[ 1,  4,  3,  2],
              [ 5,  8,  7,  6],
              [ 9, 12, 11, 10],
              [13, 16, 15, 14],
              [17, 20, 19, 18]])

```

1.3 QUESTÃO 3:

Criar uma função que recebe uma matriz como parâmetro de entrada e inverte a ordem das linhas (ou colunas) dessa matriz.

```
[11]: def trocaTudo(matriz,troca):
    if troca!='linha' and troca !='coluna':
        print("Erro! Escolha se a troca é por 'linha' ou 'coluna'!")
        return

    mat=matriz.copy()
    nlin,ncol=mat.shape
    #nlin=mat.shape[0]
    #ncol=mat.shape[1]

    if troca=='linha':
        for i in range(nlin//2): #tem que ir só até a metade, caso contrário,
            ↪ inverte o que foi feito!
            for j in range(ncol):
                aux=mat[i,j]
                mat[i,j]=mat[(nlin-i-1),j]
                mat[(nlin-i-1),j]=aux
                #mat[i,j],mat[(nlin-i-1),j]=mat[(nlin-i-1),j],mat[i,j]

    elif troca=='coluna':
        for j in range(ncol//2): #tem que ir só até a metade, caso contrário,
            ↪ inverte o que foi feito!
            for i in range(nlin):
                aux=mat[i,j]
                mat[i,j]=mat[i,(ncol-j-1)]
                mat[i,(ncol-j-1)]=aux
                #mat[i,j],mat[i,(ncol-j-1)]=mat[i,(ncol-j-1)],mat[i,j]

    return mat
```

```
[12]: #Apenas uma CURIOSIDADE sobre como criar uma matriz com os elementos em uma
        ↪ sequência escolhida.
lins=5
cols=4
tr=np.matrix(np.arange(1,lins*cols+1).reshape(lins,cols))
#tr=np.matrix([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16],[17,18,19,20]])
tr
```

```
[12]: matrix([[ 1,  2,  3,  4],
              [ 5,  6,  7,  8],
              [ 9, 10, 11, 12],
              [13, 14, 15, 16],
              [17, 18, 19, 20]])
```

```
[13]: trocaTudo(tr, 'linha')
```

```
[13]: matrix([[17, 18, 19, 20],
              [13, 14, 15, 16],
              [ 9, 10, 11, 12],
              [ 5,  6,  7,  8],
              [ 1,  2,  3,  4]])
```

```
[14]: trocaTudo(tr, 'coluna')
```

```
[14]: matrix([[ 4,  3,  2,  1],
              [ 8,  7,  6,  5],
              [12, 11, 10,  9],
              [16, 15, 14, 13],
              [20, 19, 18, 17]])
```

1.4 QUESTÃO 4:

Criar uma função para multiplicar duas matrizes, pela implementação do método de multiplicação matricial (não usar a função de multiplicação de matrizes do Numpy).

```
[15]: def matmult(a,b):
        nlinha=a.shape[0]
        ncola=a.shape[1]
        nlinb=b.shape[0]
        ncolb=b.shape[1]
        if ncola!=nlinb:
            print("Dimensões incompatíveis!")
            return nan
        nlinc=nlinha
        ncolc=ncolb
        M=np.matrix(np.zeros((nlinc,ncolc)))
        for i in range(nlinc):
            for j in range(ncolc):
                #M[i,j]=0 #não precisa pois a matriz já está zerada
                for k in range (ncola):
                    M[i,j]=M[i,j]+a[i,k]*b[k,j]
        return M
```

```
[16]: tr1=np.matrix([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16],[17,18,19,20]])
        tr2=np.matrix([[1,2],[5,6],[9,10],[13,14]])
```

```
[17]: matmult(tr1,tr2)
```

```
[17]: matrix([[ 90., 100.],
              [202., 228.],
              [314., 356.]])
```

```
[426., 484.],  
[538., 612.]])
```

```
[18]: #Vamos conferir esse resultado?  
x=tr1*tr2  
x
```

```
[18]: matrix([[ 90, 100],  
[202, 228],  
[314, 356],  
[426, 484],  
[538, 612]])
```

1.5 QUESTÃO 5:

Calcular o [determinante](#) de uma matriz usando o **método de escalonamento**.

```
[19]: def determinanteEscalonamento(matriz):  
    M=matriz.copy()  
    nlins=mat.shape[0]  
    ncols=mat.shape[1]  
    mult=1  
    for linbase in range(nlins):  
        #pega o elemento da diagonal como fator da linha base  
        fator=M[linbase,linbase]  
        mult=mult*fator  
        #divide os elementos da linha base pelo fator, pra ficar com 1 na  
        ↪ diagonal  
        for col in range(ncols):  
            M[linbase,col]=M[linbase,col]/fator  
            #substitui os elementos das linhas abaixo de forma a fazer os elementos  
            ↪ abaixo do fator ficarem com zero  
            for lin in range (linbase+1,nlins):  
                #pega como "pivô" o elemento da linha que está na coluna do fator  
                ↪ da linha base  
                pivo=M[lin,linbase]  
                for col in range(ncols):  
                    M[lin,col]=M[lin,col]-pivo*M[linbase,col]  
    return mult
```

```
[20]: mat=np.matrix([[4.,1.,2.],[3.,2.,0.],[2.,-1.,1.]])  
mat
```

```
[20]: matrix([[ 4.,  1.,  2.],  
[ 3.,  2.,  0.],  
[ 2., -1.,  1.]])
```

```
[21]: determinanteEscalonamento(mat)
```

```
[21]: -9.0
```

```
[22]: #Vamos conferir esse resultado?
print(np.linalg.det(mat))
print(round(np.linalg.det(mat),1))
```

```
-8.999999999999998
```

```
-9.0
```

1.6 QUESTÃO 6:

Calcular o **determinante** de uma matriz usando o **método dos cofatores** (recursivo).

```
[23]: def determinanteCofatores(matriz):
    M=matriz.copy()
    nlins=M.shape[0]
    ncols=M.shape[1]
    #cofs=np.zeros((1,ncols))
    if nlins==1 and ncols==1:
        return M[0,0]
    #calcula o determinante com base nos cofatores dos elementos da primeira
    ↪ linha
    det=0
    for elemento in range(ncols): #pega todos os elementos da primeira linha da
    ↪ matriz, calcula o cofator e o menor
        menor=np.zeros((nlins-1,ncols-1))
        cof=(-1)**elemento*M[0,elemento]
        for lin in range(1,nlins): #pega as linhas abaixo da primeira
            colmenor=0
            for col in range(ncols):
                if col!=elemento:
                    menor[lin-1,colmenor]=M[lin,col]
                    colmenor=colmenor+1
            det=det+cof*determinanteCofatores(menor)
    return det
```

```
[24]: mat=np.matrix([[4.,1.,2.],[3.,2.,0.],[2.,-1.,1.]])
mat
```

```
[24]: matrix([[ 4.,  1.,  2.],
             [ 3.,  2.,  0.],
             [ 2., -1.,  1.]])
```

```
[25]: determinanteCofatores(mat)
```

```
[25]: -9.0
```

```
[26]: #Vamos conferir esse resultado?
print(np.linalg.det(mat))
print(round(np.linalg.det(mat),1))
```

```
-8.999999999999998
```

```
-9.0
```

1.7 QUESTÃO 7:

Contar os pontos do boliche, as regras e um exemplo de jogo bem detalhado estão no arquivo `boliche-regras-de-pontos.docx`. Use os seguintes dados:

```
[27]: #Dados: boliche
jog1=[[10,0],[10,0],[10,0],[7,2],[8,2],[0,9],[10,0],[7,3],[9,0],[10,0],[10,8]]
jog2=[[2,5],[5,5],[10,0],[6,3],[10,0],[10,0],[9,0],[5,2],[9,1],[9,0]]
jog3=[[9,1],[10,0],[8,1],[5,4],[10,0],[10,0],[10,0],[9,1],[8,1],[8,2],[5,0]]
jog4=[[2,2],[4,1],[8,1],[6,2],[9,1],[0,10],[1,5],[2,2],[4,2],[4,1]]
```

```
[28]: def boliche(jogadas):
    jogadas=np.array(jogadas)
    n=jogadas.shape[0]
    if n==10:
        e=1
    else:
        n=10
        e=0

    #Definindo o tipo da jogada: strike, spare ou normal
    tipos=np.zeros(n+1).astype(str) #adicionar mais um índice para ajudar a
    ↪ calcular o bônus
    pontosbase=np.zeros(n)
    for j in range(n):
        if jogadas[j,0]==10:
            tipos[j]='strike'
        elif jogadas[j].sum()==10:
            tipos[j]='spare'
        else:
            tipos[j]='normal'

    #Calculando os pontos-base
    pontosbase[j]=jogadas[j].sum()

    #Calculando o bônus
    bonus=np.zeros(n+e)
    for t in range(n+e):
        if tipos[t]=='strike':
```



```

        if tipos[t+1]=='strike':
            pinosderrubados=jogadas[t+1,0]+jogadas[t+1,1]+jogadas[t+2,0]
        else:
            pinosderrubados=jogadas[t+1,0]+jogadas[t+1,1]
    elif tipos[t]=='spare':
        pinosderrubados=jogadas[t+1,0]
    else:
        pinosderrubados=0
    bonus[t]=pinosderrubados

    pontosjogada=np.zeros(n)
    for i in range(n):
        pontos=pontosbase[i]+bonus[i]
        pontosjogada[i]=pontos

    return int(pontosjogada.cumsum()[-1])

```

```

[29]: print('Jogador 1: ', boliche(jog1))
      print('Jogador 2: ', boliche(jog2))
      print('Jogador 3: ', boliche(jog3))
      print('Jogador 4: ', boliche(jog4))

```

```

Jogador 1: 180
Jogador 2: 147
Jogador 3: 178
Jogador 4: 68

```

1.8 QUESTÃO 8:

Campo minado: Defina o tamanho n do quadro do jogo e a quantidade p de bombas no quadro. Crie uma função que recebe como parâmetro de entrada uma matriz $p \times 2$, com a lista das posições onde estão localizadas as bombas (em cada linha, a primeira coluna contém o índice da linha e a segunda coluna o índice da coluna onde está localizada a bomba); e retorna como saída a matriz de vizinhança, de dimensão $n \times n$, contendo, em cada célula, a quantidade de bombas existentes nas células vizinhas.

As células onde estão localizadas as bombas devem ser preenchidas com NaN (not a number - `np.nan`) e as células que não têm bombas nas vizinhanças devem ser preenchidas com zeros.

```

[30]: #Matriz com a localização das bombas
      bombas=np.matrix([[0,2],[0,3],[1,6],[2,2],[2,7],[3,0],[3,4],[4,6],[5,2],[6,5]])
      bombas

```

```

[30]: matrix([[0, 2],
              [0, 3],
              [1, 6],
              [2, 2],
              [2, 7],

```

```
[3, 0],
[3, 4],
[4, 6],
[5, 2],
[6, 5]])
```

```
[31]: def campominado(bombas,tamanho1,tamanho2):
        campo=np.zeros([tamanho1,tamanho2])
        nbombas=bombas.shape[0]
        for b in range(nbombas):
            #marca a célula da bomba com "nan" (not a number)
            campo[bombas[b,0],bombas[b,1]]=np.nan
            #aumenta a quantidade de bombas nas células vizinhas
            i=bombas[b,0]
            j=bombas[b,1]
            x1,x2,y1,y2=0,0,0,0
            #esquerda
            if j>0:
                x1=-1
            #direita
            if j<tamanho2-1:
                x2=2
            #em cima
            if i>0:
                y1=-1
            #em baixo
            if i<tamanho1-1:
                y2=2
            campo[i+y1:i+y2,j+x1:j+x2]+=1
        return campo
```

```
[32]: campominado(bombas,8,8)
```

```
[32]: array([[ 0.,  1., nan, nan,  1.,  1.,  1.,  1.],
             [ 0.,  2.,  3.,  3.,  1.,  1., nan,  1.],
             [ 1.,  2., nan,  2.,  1.,  2.,  2., nan],
             [nan,  2.,  1.,  2., nan,  2.,  2.,  1.],
             [ 1.,  2.,  1.,  2.,  1.,  2., nan,  1.],
             [ 0.,  1., nan,  1.,  1.,  2.,  2.,  1.],
             [ 0.,  1.,  1.,  1.,  1., nan,  1.,  0.],
             [ 0.,  0.,  0.,  0.,  1.,  1.,  1.,  0.]])
```

1.9 QUESTÃO 9:

Uma das tarefas computacionais importantes é a localização de padrões de repetições em conjuntos de dados. Sua tarefa nesta questão é desenvolver um código para localizar e contar as ocorrências de uma sequência de valores dentro de uma matriz.

- a) Criar uma função que recebe como parâmetros de entrada (i) uma matriz de dados; e (ii) um valor a ser procurado, e retorna como parâmetros de saída (i) a quantidade de vezes que este valor aparece na matriz, e (ii) uma lista com as posições (linha coluna) onde esses valores são encontrados.

```
[33]: dados=np.matrix(pd.read_excel('ocorrencias.xlsx',header=None))
seq1=np.matrix(pd.read_excel('ocorrencias.xlsx',header=None,sheet_name="seq1"))
seq2=np.matrix(pd.read_excel('ocorrencias.xlsx',header=None,sheet_name="seq2"))
seq3=np.matrix(pd.read_excel('ocorrencias.xlsx',header=None,sheet_name="seq3"))
```

```
[34]: def contaocorr(matriz,valor):
    mat=matriz.copy()
    nlins=mat.shape[0]
    ncols=mat.shape[1]
    #conta quantas vezes o "valor" aparece na matriz "mat"
    cont=0
    lista=[]
    for i in range(nlins):
        for j in range(ncols):
            if mat[i,j]==valor:
                cont=cont+1
                lista.append([i,j])
    return cont,lista
```

```
[35]: print(contaocorr(dados,20)[0])
print()
print(contaocorr(dados,20)[1])
```

119

```
[[5, 14], [10, 29], [13, 29], [21, 23], [29, 30], [31, 5], [38, 2], [40, 12],
[52, 3], [53, 1], [53, 40], [60, 15], [61, 20], [64, 25], [67, 21], [72, 9],
[74, 37], [78, 45], [82, 6], [82, 46], [84, 26], [85, 25], [85, 40], [92, 10],
[92, 11], [94, 44], [96, 18], [98, 43], [100, 8], [105, 38], [110, 6], [114,
29], [116, 45], [118, 21], [119, 49], [121, 33], [125, 15], [131, 2], [136, 5],
[138, 43], [138, 47], [140, 31], [148, 7], [157, 47], [164, 38], [166, 12],
[166, 14], [172, 17], [176, 43], [182, 37], [191, 18], [191, 34], [195, 9],
[196, 23], [198, 13], [198, 32], [200, 37], [207, 8], [216, 24], [225, 28],
[225, 43], [230, 22], [234, 8], [236, 20], [237, 33], [260, 26], [275, 14],
[285, 30], [286, 1], [288, 49], [294, 13], [301, 12], [309, 34], [319, 13],
[324, 24], [325, 45], [330, 26], [331, 23], [345, 13], [346, 49], [347, 42],
[347, 47], [354, 2], [355, 14], [359, 28], [360, 15], [363, 14], [368, 6], [370,
14], [371, 44], [375, 39], [377, 37], [382, 15], [384, 41], [399, 35], [404,
14], [407, 25], [414, 2], [418, 35], [420, 28], [422, 18], [425, 30], [427, 35],
[427, 44], [428, 2], [435, 45], [442, 6], [444, 14], [445, 16], [456, 22], [458,
6], [468, 19], [471, 15], [473, 48], [474, 13], [476, 24], [482, 27], [483, 36],
[491, 0]]
```

- b) Criar uma função que recebe como parâmetros de entrada uma matriz de dados e uma sequência de valores (array ou lista), e localiza a sequência na matriz, retornando a lista com as posições (i, j) onde a sequência foi encontrada.

Por exemplo, teremos o seguinte resultado quando procuramos a sequência $[1\ 2\ 3]$ na matriz abaixo:

8	10	14	13	3
2	1	2	3	14
3	6	2	11	11
5	12	1	2	3
7	15	4	9	3
1	2	3	4	7

N.Rep: 3

1	1
3	2
5	0

Você pode testar sua função com os dados disponíveis na planilha dados e com os vetores seq1, seq2 e seq3, do arquivo `ocorrencias.xls`. Os resultados esperados são os seguintes:

Seq1

N.Rep: 3

14	3
66	25
115	45

Seq2

N.Rep: 5

4	19
19	2
20	37
281	34
439	16

Seq3

N.Rep: 2

10	40
36	4

```
[36]: def contaocorr2(matriz,seq):  
    mat=matriz.copy()  
    nlin=mat.shape[0]  
    ncol=mat.shape[1]  
    #conta quantas vezes a sequência "seq" aparece na "matriz", e identifica as  
    ↳posições das ocorrências  
    cont=0  
    lista=[]  
    tamanho=seq.shape[1]  
    primeiro=seq[0,0]  
    for i in range(mat.shape[0]): #todas as linhas da matriz  
        for j in range(ncol-tamanho+1): #todas as colunas, até a última possível  
            if mat[i,j]==primeiro: #se encontrou o primeiro elemento da  
            ↳sequência, vai tentar achar os demais  
                achou=1  
                ind=1  
                while achou==1 and ind < tamanho:  
                    if mat[i,j+ind]!=seq[0,ind]:  
                        achou=0  
                    ind=ind+1  
                if achou==1:  
                    cont=cont+1
```

```

        lista.append([i,j])
    return cont,lista

```

```

[37]: M=np.
      ↪matrix([[8,10,14,13,3],[2,1,2,3,14],[3,6,2,11,11],[5,12,1,2,3],[7,15,4,9,3],[1,2,3,4,7]])
      M

```

```

[37]: matrix([[ 8, 10, 14, 13,  3],
              [ 2,  1,  2,  3, 14],
              [ 3,  6,  2, 11, 11],
              [ 5, 12,  1,  2,  3],
              [ 7, 15,  4,  9,  3],
              [ 1,  2,  3,  4,  7]])

```

```

[38]: seq=np.matrix([1,2,3])
      seq

```

```

[38]: matrix([[1, 2, 3]])

```

```

[39]: contaocorr2(M,seq)

```

```

[39]: (3, [[1, 1], [3, 2], [5, 0]])

```

```

[40]: contaocorr2(dados,seq1)

```

```

[40]: (3, [[14, 3], [66, 25], [115, 45]])

```

```

[41]: contaocorr2(dados,seq2)

```

```

[41]: (5, [[4, 19], [19, 2], [20, 37], [281, 34], [439, 16]])

```

```

[42]: contaocorr2(dados,seq3)

```

```

[42]: (2, [[10, 40], [36, 4]])

```

1.10 QUESTÃO 10:

Troca-troca de figurinhas: Criar uma função que recebe dois pares de listas, representando as figurinhas de dois jogadores: para cada jogador, passe a lista de figurinhas que ele precisa e a lista de figurinhas que ele tem duplicatas. Retorne com a lista de trocas que esses dois jogadores podem fazer entre si.

```

[43]: def trocafigurinhas(jogador1, jogador2):
      precisa1=jogador1[0]
      precisa2=jogador2[0]
      duplicatas1=jogador1[1]
      duplicatas2=jogador2[1]

```

```

trocas1=[]
trocas2=[]
if len(duplicatas1)>len(duplicatas2):
    for fig1 in duplicatas1:
        for fig2 in precisa2:
            if fig1==fig2:
                trocas1.append(fig1)
    for fig2 in duplicatas2:
        for fig1 in precisa1:
            if fig1==fig2:
                trocas2.append(fig1)
else:
    for fig2 in duplicatas2:
        for fig1 in precisa1:
            if fig1==fig2:
                trocas2.append(fig1)
    for fig1 in duplicatas1:
        for fig2 in precisa2:
            if fig1==fig2:
                trocas1.append(fig1)
trocas=[]
maximo=max(len(trocas1),len(trocas2))
for i in range(len(trocas2)):
    trocas.append([trocas1[i],trocas2[i]])

return trocas

```

```

[44]: jogador1=[[2,7,8,9,10],[1,3,4,6]]
      jogador2=[[1,4,6,8],[2,3,5,7]]
      trocafigurinhas(jogador1,jogador2)

```

```

[44]: [[1, 2], [4, 7]]

```

```

[45]: jogador1=[[2,7,8,9,10],[1,3,4,6]]
      jogador2=[[1,4,6,8],[2,3,5,7,10]]
      trocafigurinhas(jogador1,jogador2)

```

```

[45]: [[1, 2], [4, 7], [6, 10]]

```