main

Controla o fluxo de execução e printa a matriz ao final. Quando a matriz fica com somente dois “nodos” não é possível realizar as operações então os nodos são só juntados. Tivemos que multiplicar a matriz por -1 pois na questão estávamos recebendo uma matriz de scores.

```
if __name__ == '__main__':
    global mat
    print ("0 - Digitar matriz\n1 - Utilizar matriz exemplo")
    op = input()
    mat,names = inputF(op)
    mat = mat*-1
    while(len(mat)>2):
        ui = uiCalculation()
        dij = smallestDij(ui)
        v = valueBranch(dij,ui)
        joinTree(names[dij[1]],names[dij[2]],v)
        newMatrix(dij,v)

    joinTree("end",names[1],[0,mat[0][0]])
    print(tree)
```

inputF

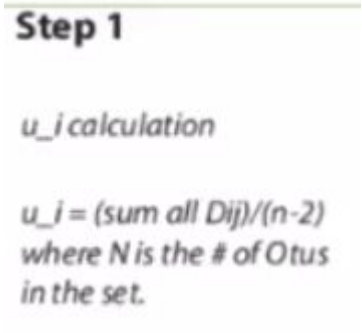
Função que retorna a matriz de entrada e nomes dos nodos. Há a opção de receber o input do teclado, ou usar a matriz de exemplo.

```
def inputF(op):
    mat = []
    if not op:
        names = input()
        names = names.split("\t")
        print (names)
        for i in range (len(names)):
            linha = input()
            mat[i] = linha.split("\t")
    else:
        mat = [[0,1,3,2.8],[1,0,2.8,2.7],[3,2.8,0,0.15],[2.8,2.7,0.15,0]]#exemplo
        names = ["b", "r", "c", "g"]#exemplo
    return [mat,names]
```

Step 1 - uiCalculation

Calcula todos u_i,somando todas distancias até um nodo e dividindo pela quantidade de nodos-2.

```
def uiCalculation():
    tam = len(mat)
    global ui
    ui= np.zeros(tam)
    for x in range(tam):
        for y in range(tam):
            ui[x] += mat[x][y]
              ui[x] = ui[x]/(tam-2)
    return ui
```



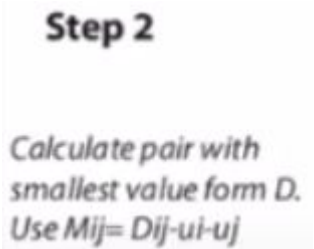
Step 2 - smallestDij

```
def smallestDij(ui):
    tam = len(mat)
    x = y = start =0

    min = [mat[0][1] - ui[0] -ui[1],0,1]
    while (x<tam):
        start+=1
        y=start
        while (y<tam):
            d = mat[x][y] - ui[x] -ui[y]
            if (d<min[0]):
                min = [d,x,y]

            y+=1
        x+=1
    return min
```

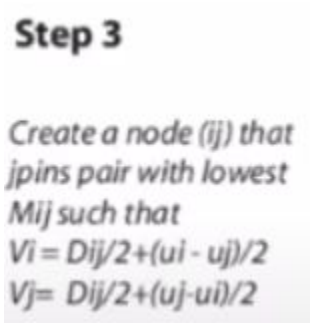
Calcula a menor distância, usando os ui calculados no passo anterior.



Step 3 - valueBranch

```
def valueBranch(dij,ui):
    x = dij[1]
    y = dij[2]
    dist = mat[x][y]
    v = np.zeros(2)
    v[0] = abs((dist + (ui[x]-ui[y]))/2)
    v[1] = abs((dist + (ui[y]-ui[x]))/2)
    return v
```

Calcula o valor de distância de cada branch, usando o menor valor que foi calculado por smallestDij



Step 4 -joinTree

Junta os nodos na árvore no formato Newick tree, usando o valores de distância dos branches, conforme os casos descritos com os comentários.

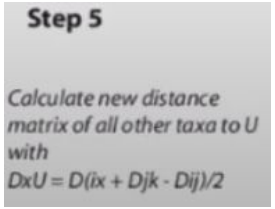
```
def joinTree(x,y,v):
    global tree
    findx = tree.find(x)
    findy = tree.find(y)
    if (x=="end"): #se é o último nodo a ser inserido
        tree = tree[:len(tree)-1]+ "," +y +":0" + ")"
    elif (findx !=-1 and findy!=-1):#se os dois estão na árvore
        tree =  tree.replace(x+",","")
        tree = tree.replace(y+",","")
        tree = "("+x+","+y+"),"+tree
    elif(findx !=-1): #se x está
        tree =  tree[0:findx] + "("+ tree[findx:findx+len(x)]+":"+ str(round(v[0],4)) + "," +y +":"+
str(round(v[1],4))+ ")" + tree[findx+len(x):len(tree)]
    elif(findy !=-1): #se y está
        tree =  tree[0:findy] + "("+ tree[findx:findy+len(y)]+":"+ str(round(v[1],4))  + "," +x +":"+
str(round(v[0],4))+ ")"+ tree[findy+len(x):len(tree)]
    else: #se nenhum está
        tree = "("+ x+":"+ str(round(v[0],4)) + ","+ y+":"+ str(round(v[1],4)) + ")"
```

Step 5 - newMatrix

Calcula a nova matriz e atualiza os nomes dos nodos. Para isso, calcula as novas distâncias relativas ao novo nodo e usa parte da matriz antiga.

```
def newMatrix(dij,v):
    global mat
    x = dij[1]
    y = dij[2]
    tam = len(mat)

    #altera os nomes dos nodos
    newName = "(" + names[x]+":"+
str(round(v[0],4))+","+names[y]+":"+
str(round(v[1],4))+")"
    name1 = names[x]
    name2 = names[y]
    names.remove(name1)
    names.remove(name2)
    names.insert(0,newName)
```




```
    #calcula as novas distancias
    nDist = np.zeros(len(mat))
    for i in range (len(mat)):
        nDist[i] = (mat[x][i] +
mat[y][i]-mat[x][y])/2
    nDist = np.delete(nDist,(x,y),0)
    #deleta as colunas/linhas que
serão descartadas da matrix antiga
    mat = np.delete(mat,(x,y),0)
    mat = np.delete(mat,(x,y),1)
    #calcula a nova matriz, usando
parte da antiga e as novas
distancias nDist
    nMat= np.zeros((tam-1,tam-1))
    nMat[1:tam-1,0] = nDist
    nMat[0,1:tam-1] = nDist
    nMat[1:tam-1,1:tam-1] = mat
    mat = nMat
```

Resultado

Resultado com a matriz

	L. braziliensis	T. rangeli	T. cruzi	T. gambiae
L. braziliensis	0.000	0.010	0.300	0.280
T. rangeli	0.010	0.000	0.280	0.270
T. cruzi	0.300	0.280	0.000	0.015
T. gambiae	0.280	0.270	0.015	0.000

((brazilienses:0.0125,rangeli:0.0025):0.27,cruzi:0.015,gambiae:0)