UNIQUE DISTANCING PUZZLE

https://www.think-maths.co.uk/uniquedistance

Is it always possible to put n counters on an n x n grid, such that no two counters are the same distance apart? Here distance is the usual Euclidean distance.

Puzzle for Submission: Can you place 6 counters on a 6x6 grid such that the distance between each counter is different?

There are 36 choose 6 = 1947792 possible board configurations. So a solution using an exhaustive seemed a reasonable approach. Using the Python code below, 16 board configurations were found. Apparently up to rotations and reflections there are two fundamental solutions:

```
# _ # _ _ #
_ _ _ _ _ _
_ _ _ _ _ _
_ _ _ # _ _
_ _ _ _ # _
_ _ _ _ # _
```

```
# # _ _ _ _
_ _ _ # _ _
_ _ _ _ _ _
_ _ _ _ _ #
_ _ _ _ _ _
_ _ # _ _ #
```

A pigeon hole principle argument shows that for sufficiently large boards (n>144) it is not possible possible to put n counters on an n x n grid, such that no two counters are the same distance

```
"""

UNIQUE DISTANCING PUZZLE

https://www.think-maths.co.uk/uniquedistance

Can you place 6 counters on a 6x6 grid such that

the distance between each counter is different?

Here distance is the usual Euclidean

a 3X3 Case solution is below
```

```
 0 0 1
 1 1 0
 0 0 0
"""
import math
from itertools import combinations
#Displays grid
def printGrid(grid):
    for i in range(len(grid)):
        for j in range(len(grid)):
            print(grid[i][j], end =" ")
        print()
    print()
#Euclidean distance
def dist(x1,y1,x2,y2):
    return math.sqrt((x1-x2)**2+(y1-y2)**2)
#Creates 6x6 grid of points
g = []
for i in range(6):
    for j in range(6):
        g.append((i,j))
```

```
#Create list of all possible 6 element combinations of points

possibleSix = list(combinations(g, 6))


#Exhaustive search through each 6 element combination

#Valid combinations need all 15 possible distance pairs

#to be different. Use a Python set to check each combination

for p in possibleSix:

    l = []

    d = set () #distance set

    #Main loop through all combinations

    for i in range(len(p)):

        #Check a particular combination

        for j in range(len(p)):

            if (i<j):

                l.append((p[i],p[j]))

                for item in l:

                    x1=((item)[0])[0]

                    x2=((item)[1])[0]

                    y1=((item)[0])[1]

                    y2=((item)[1])[1]

                    d.add(dist(x1,y1,x2,y2))
```

```
#if set has 15 elements print

    if(len(d)==15):

        print(p)

#All 16 valid combinations

ans = [((0, 0), (0, 1), (1, 3), (3, 5), (5, 2), (5, 5)),

((0, 0), (0, 2), (0, 5), (3, 3), (4, 4), (5, 4)),

((0, 0), (0, 2), (2, 4), (3, 0), (4, 5), (5, 5)),

((0, 0), (0, 3), (0, 5), (3, 2), (4, 1), (5, 1)),

((0, 0), (0, 3), (2, 0), (4, 2), (5, 4), (5, 5)),

((0, 0), (1, 0), (2, 5), (3, 1), (5, 3), (5, 5)),

((0, 0), (1, 4), (1, 5), (2, 3), (3, 0), (5, 0)),

((0, 0), (2, 0), (3, 3), (4, 4), (4, 5), (5, 0)),

((0, 1), (1, 1), (2, 2), (5, 0), (5, 3), (5, 5)),

((0, 2), (0, 5), (2, 5), (4, 3), (5, 0), (5, 1)),

((0, 3), (0, 5), (2, 1), (3, 5), (4, 0), (5, 0)),

((0, 4), (0, 5), (1, 2), (3, 0), (5, 0), (5, 3)),

((0, 4), (1, 4), (2, 3), (5, 0), (5, 2), (5, 5)),

((0, 5), (1, 0), (1, 1), (2, 2), (3, 5), (5, 5)),

((0, 5), (1, 5), (2, 0), (3, 4), (5, 0), (5, 2)),

((0, 5), (2, 5), (3, 2), (4, 0), (4, 1), (5, 5))]
```

```
#Displays valid combinations in grid form

for item in ans:

    g= [['_' for i in range(6)] for j in range(6)]

    for t in item:

        r = t[0]

        c = t[1]

        g[r][c] = '#'

    printGrid(g)

    print()
```