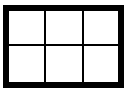
AI - 10 Nov. 2018

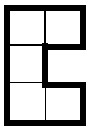
Rectangular Tanagrams

The second type of puzzle that you should solve, using a second script, are rectangular Tanagrams. Rectangular Tanagrams are polygons where each pair of boundary segments are either parallel or perpendicular. With rectangular Tanagrams, the goal is to put the pieces into a containing rectangle, but it is no longer assumed that the tanagrams must completely cover the containing rectangle. In other words, it is easily possible that there are leftover holes at the end – this will depend on what is given.

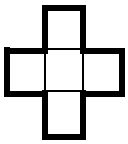
Since the individual pieces are probably not rectangular, we must specify them explicitly. The way to do this is to declare a bounding rectangle and then give the explicit layout within that bounding rectangle where a `y` represents part of the tanagram piece and a period represents the absence of the tanagram piece in question. Some common examples are:



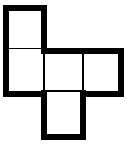
Rectangle
`2x3 yyyyyy`



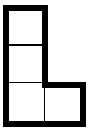
C shape
`3x2 yyy.yy`



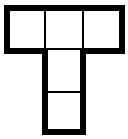
Cross
`3x3 .y.yyy.y.`



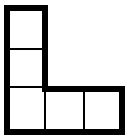
Pentomino 3
`3x3 y..yyy.y.`



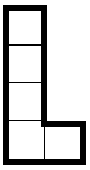
L shape short
`3x2 y.y.yy`



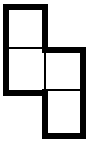
T shape
`3x3 yyy.y..y.`



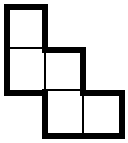
L shape wide
`3x3 y..y..yyy`



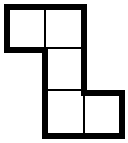
L shape tall
`4x2 y.y.y.yy`



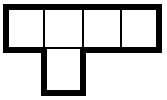
Jog
`3x2 y.yy.y`



W shape
`3x3 y..yy..yy`



Z shape
`3x3 yy..y..yy`



Pentomino 4
`2x4 yyyy.y..`

Note that tanagram shapes may overlap as long as there is no overlapped position in the containing rectangle with more than one `y` at that position. Also, the tanagram shapes in the rightmost column above, and the short L, may be placed in any of 8 positions (four rotations, and a flip for each rotation).

Therefore, your code should account for rotation of pieces by 90 degrees and also reflections. This is the last part that you should implement in your code, because even though these two functions are one liners, packaging them up appropriately, and developing the one liner can be a bit tricky.

Here are some basic examples:

`tanagram.py 2x3 2x3 yyy..y`	Most basic
`tanagram.py 2x4 2x3 yyy..y 2x3 y..yyy`	Multiple pieces
`tanagram.py 2x4 2x3 yyy..y 2x3 yyy..y`	Must use 180º rotation
`tanagram.py 4x2 2x3 yyy..y 2x3 yyy..y`	Must use 90º rotation
`tanagram.py 2x4 2x3 yyy..y 2x3 yyyy..`	Must use a flip
`tanagram.py 4x2 2x3 yyy..y 2x3 yyy..y`	Leaves a hole in the middle

Intermediate examples:

`tanagram.py 6 6 3x3 .y.yyyy.. 3x3 .y.yyyy.. 4x2 y.yyy.y. 3x3 yyy.y..y. 3x3 yy..yy..y 3x3 ..y..yyyy`	No rotations or flips necessary
`tanagram.py 6 6 3x3 .y.yyyy.. 3x3 .y.yyyy.. 4x2 y.yyy.y. 3x3 yyy.y..y. 3x3 yy..yy..y 3x3 yyyy..y..`	At least one rotation or flip is necessary

There is generally more than one solution to these problems. Here is a possible output for the intermediate example (they are the same puzzle):

```
Solution found:
.ABBB.
AAABC.
ADDBCC
.EDDCF
EEEDCF
E..FFF
```

You may think of a different / better way. If so, please let me know on Monday. For the scheme in this example, each time a tanagram piece was placed, that is when it received its letter identifier. An alternate scheme, which would probably make debugging easier, is to decide ahead of time which piece gets which identifier (for example, the first input piece gets A, the second piece in the input gets B, and so on). Another good place for a lookup table.