

Judicious brute force

There is a class of puzzles where each position of the puzzle is to be filled in with a symbol. This could be as simple as a 0/1 symbol (such as in the n -queens problem where a 0 means no queen at that position and a 1 means there is a queen at that position), or numbers (such as 1-9 in most Sudoku problems), or arbitrary labels (such as 0-9 and A-F in 16 x 16 sudokus). Often, as with a standard Sudoku puzzle, the numbers are acting as only labels, meaning that the numeric qualities of the number are not being used. In those cases, letters or other symbols could just as easily be used.

A possible (horrible) approach to solving this type of puzzle might be

```
def bruteForce(pzl):
    # returns a solved pzl or the empty string on failure
    if pzl is completely filled out:
        return "" if isInvalid(pzl) else pzl

    find a setOfChoices that is collectively exhaustive
    for each possibleChoice in the setOfChoices:
        subPzl = pzl with possibleChoice applied
        bF = bruteForce(subPzl)
        if bF: return bF
    return ""
```

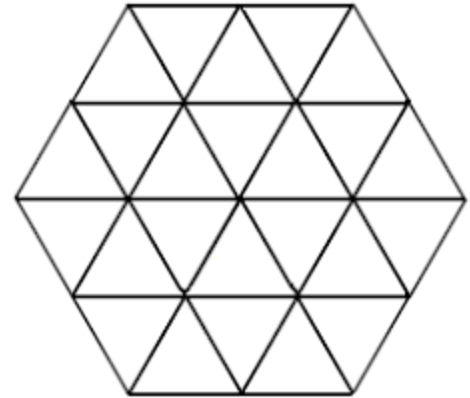
The reason this is so horrible is that it fills each puzzle out completely before testing. For all but the most trivial of problems, the routine will hang because there are way too many possibilities to recurse through. However, we can gain considerable traction with only a slight modification. By testing the validity of a potential solution at each step of the process, it is the hope that large subtrees of invalid possibilities may be immediately pruned. For example, in the n -queens problem, if two queens attacked each other, there is no reason to continue filling in the puzzle since it's already unsolvable. The following is what was shown in class as our starting out point for the month:

```
def bruteForce(pzl):
    # returns a solved pzl or the empty string on failure
    if isInvalid(pzl): return ""
    if isSolved(pzl): return pzl

    find a setOfChoices that is collectively exhaustive
    for each possibleChoice in the setOfChoices:
        subPzl = pzl with possibleChoice applied
        bF = bruteForce(subPzl)
        if bF: return bF
    return ""
```

Your homework is to use the above *judicious* bruteForce code (or small variation on it) to solve the following questions:

You are given an inner hexagon (made up of 6 small triangles), and around each of the 6 outer vertices of the hexagon you construct another hexagon to wind up with a total of 7 complete hexagons utilizing 24 of the small triangles in total. In other words, the outer perimeter has 18 triangles.



In addition, for each of the three main directions for this amalgamation of triangles, there are 4 rows of either 5 or 7 triangles. That is, there are 12 distinct rows of triangles.

- Q1) Is there a way to label each small triangle with an integer from 1-6 where none of the seven hexagons has two triangles with the same number?
- Q2) Is there a way to label each small triangle with an integer from 1-7 where none of the seven hexagons and none of the twelve rows of triangles has two triangles with the same number?

If the answer to either question is yes, you should provide a solution.

Specifics: You are to write a command line script that takes up to two arguments. If one of the arguments is a B, then you should attempt to solve Q2. Otherwise, attempt to solve Q1. This argument is optional. The other argument is required, namely a length 24 string consisting of the symbols 1-6 and the period. Your script is to attempt to assign symbols in place of the periods so as to satisfy the problem's stipulations. The arguments could come in either order.

One question that must be addressed is how the input string maps to the image above. Perhaps the answer can be provided by the following mapping: ACDEFGHIJKLMNOPQRSTUVWXYZ =>

A B C D E	(5 triangles)
F G H I J K L	(7 triangles)
M N O P Q R S	(7 triangles)
T U V W X	(5 triangles)

Your output should either be:

No solution possible

or an output a single string representing the solution, according to the same scheme just depicted, **along with** the second representation taking up four rows

Considerations:

Your program is not expected to draw triangles or hexagons. However, it must have some way of identifying each of the triangles. So, you will have to devise some indexing scheme to decide which is the 0 triangle, which is triangle with index 1, and so forth. As long as each triangle has a unique index, it would be a valid indexing (you could even give each triangle a name, like "Fred" and "Bill" and "Martha", but it's not clear to me what purpose this would serve).

What is important is what sets of triangles constitute a hexagon, or a row of triangles. If you make a dictionary of each such set, then your code should run just fine.

Use a period (.) to represent an unfilled position in the puzzle. This is because _ and spaces tend to merge when displayed while the periods will stay distinct.

In your code, should I look, I expect to see `isInvalid(pzl)` and `bruteForce(pzl)` explicitly appear as functions. You should be prepared to run your code at the **start of class** on **Tuesday / Wednesday**.

Here is an example puzzle:

```
  1 2 3 4 5
6 . . . . . 6
1 . . . . . 1
  5 4 3 2 1
```

where the top row represents the first 5 triangles going across, the 2nd row represents the next 7, and so on. If the above were to be input as a single string, it would look like:

```
hexy.py 123456.....61.....154321 A
```

with at most 10 characters to be filled out (or a determination made that no solution is possible). The original Q1 and Q2 just want a string of 24 dots.