randomized recursive rendering of a positive integer using NATO number flags

HW5 is about creative design, turtle graphics, recursion, building functions, and randomization.

HW5 syntax tested

design of functions, turtle graphics, recursion, randomization, and some of the earlier syntax

HW5 style constraints

- each line should contain no more than 80 characters
- functions should contain docstrings (using our 103 style)

Deliverables

hw5_19fa103.py

The HW5 problem

You will draw a positive integer using number flags (of the NATO flavour). International maritime signal flags may be used to represent digits: see the **Wikipedia page** and the section on `Number flags`. **We will use the NATO flags in the left column.**

For example, if `nato(735)` is called, turtle graphics should be used to render an image as shown here:

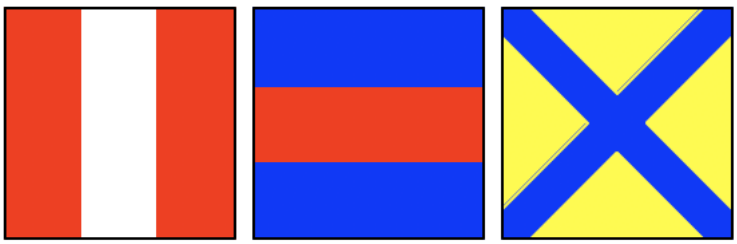

Figure 1: rendering 735 using maritime flags

You will use a randomized recursive solution, as described below.

Randomization

You will use randomization to vary how the size and position of the flags. So each time the code is executed, it will look slightly different. This randomization is implemented as follows. There are 3 random elements: the location of the bottom left corner of the first (leftmost) flag, the gap between consecutive flags, and the size of each flag (all are the same size). In particular, the bottom left corner of the first flag is drawn at a random point in the rectilinear square bounded by (-400,-100) and (-350,0). Each flag is of the same width w, where w is chosen randomly between 50 and 100 (inclusive). The gap between consecutive flags is a random number between 1 and 10 (inclusive). Since these random numbers determine the size and position of the flags, each time you run the code you should see a slightly different drawing.

The use of randomization is my way of encouraging the use of variables over magic numbers. A statement like 'forward(40)' uses a magic number 40. A statement like 'forward(w/2)' where 'w' has a semantic meaning of width, is much better: it documents your thought, and it is easily generalized to different widths. Since you won't know the exact measurements until you run the code, and these measurements will change with each execution, you must be using variables to represent distances and coordinates.

Recursion

The sole purpose of the driver function `nato` is to randomly choose the bottom left corner, width, and gap, then call the main function `natoRecur`. `natoRecur` uses recursion (base case and recursive call: see lecture notes and code examples). There is a natural base case and a natural recursive call (just one). Two of the three parameters to natoRecur will change in the recursive call: it is your job to find out how you want them to change, and how to calculate these new values for the parameters.

Helper functions

You should implement several helper functions, functions that are called by natoRecur to help it divide and conquer its task. 10 obvious helper functions are the functions that draw the 10 digit flags (I suggest the function names draw0, draw1, ...) Many of these digit functions will be similar to each other. The digit functions should themselves have helper functions, since they contain repeatable elements, such as rectangles or triangles or other shapes (e.g., crosses).

Drawing style

Please draw a black border around each flag (1 pixel wide). This is useful for distinguishing certain flags, such as the 0 flag.

0 flag

I'd suggest working on the 0 flag last (even though we are computer scientists). So I'd suggest working on the other flags first, to get your feet wet. You can test on numbers that avoid 0 too, until you get those working (e.g., set n = '123456789'). The 6 flag is also more challenging.

Interpretation of distances in the maritime flags

Here is the analysis of the flags that you should use to drive your decision on distances.
Linear interpolation, anyone?

0 flag the circumscribing square of the plus sign has width w/5 and the distance of the bottom left plus sign's circumscribing square from the outer square is w/15

4 and 5 flags vertical bands on far left and far right are 3/4 w; tip of red triangles are 3/4 of the way to the center

6 flag challenge: what is the width of the white band (where it hits the bottom of the square)?

1/2/3/7/8/9 obvious

Incomplete answer

If you cannot implement all 10 digit flags, you may replace those digits by empty squares as you encounter them in the integer you are given. For example, if you only implement 0-5, you would draw the integer 3672 as if it were 3-flag empty-square empty-square 2-flag. You will only receive credit for the flags you implement (see below).

Grading

grading graded by hand, by running your code repeatedly (with a different result each time since it is randomized), then evaluating your code for style

base points available

overall solution works (perhaps with some missing digits) +5
helper functions (with docstrings) +1 for each digit, +1 for all other fns combined
total available 16

base grade

A+ 16
A 15
B 11 (requires at least 6 digits)
C 7 (requires at least 2 digits)

adjustments (anticipated)

recursion adjustment if an iterative solution is used rather than a recursion one in natoRecur, -1 letter grade
randomization adjustment if the position/size/gap are not all randomized, -1 letter grade
style adjustment
particularly elegant answer: +1 letter grade
ugly or awkward answer (e.g., confusing code; poor breakdown into functions): -1 letter grade

challenge

You can almost use a different version of natoRecur, using an integer for the first parameter rather than a string. Explain how this would work in general, then explain why this breaks down on 0 digits.