Com S 127x
Fall 2016
Assignment 3

Due Date: Tuesday, October 18, 11:59 pm (midnight)

"Late" deadline (25% penalty): Wednesday, October 19, 11:59 pm

General information

This assignment is to be done on your own. See the Academic Dishonesty policy in the syllabus for details.

If you need help, see your instructor or one of the TAs. Lots of help is also available through the Piazza discussions.

Please start the assignment as soon as possible and get your questions answered right away.

Introduction and review

There are two problems, both involving turtles. The purpose of the assignment is to practice with the ideas of *incremental development* and *procedural decomposition*, similar to the example we did in class on Friday of drawing a "flower" by adding a petal-like shape at each vertex of a polygon. You should review the code we wrote in class, http://web.cs.iastate.edu/~smkautz/cs127f16/examples/week7/arc_experiment.py

As a brief review of what it's all about, the idea is to start with the questions:

Can we solve part of the problem? Can we solve a related, simpler problem?

The questions above lead us to further understanding of the problem, so we can see how to solve the larger problem. In some cases, if we are lucky, we can also take our solution for part of the problem and code it as a Python function that can be *composed* with other functions to develop a solution to the larger problem. Figuring out what these smaller pieces are in procedural decomposition. It goes hand in hand with the problem-solving strategy based on the two questions above.

So when we did this on Friday, the ideas above led us to the further questions:

- *Can we draw just the n-sided polygon?*
    - Yes, this is a problem we have solved already (in lab)
- *Can we draw just one petal?*
    - *Can we draw just one arc of a petal?*
        - Start with a concrete example: draw 10 segments, turn left a few degrees between each segment.
        - To better re-use the code for drawing an arc, we define a *function* that draws an arc with a given turtle. (This is an example of procedural decomposition.)
    - To draw the petal, we just draw two arcs. We just needed to make sure we are clear about what direction the turtle is facing after drawing one of the arcs.
- *Now, can we orient the petal at the correct angle at each vertex of the polygon?*
    - We first solve the simpler problem: can we draw a straight line out from each vertex, in the direction we want the petal to go?
        - To solve this, we start with the concrete instance: can we do it for a square?

Problem 1

a) The function **`draw_arc`** in the sample file **`arc_experiment.py`** always draws an arc of 10 line segments with a total turn angle of 40 degrees (4 degrees per line segment). Each line segment has length 10. *Generalize* this function as follows:

```
def draw_arc(t, angle, num_segments, length)
```

where **`t`** is the given turtle
`angle` is the total turn angle (in this case 40 degrees)
`num_segments` is the number of segments (in this case 10)
`length` is the total length of the segments (in this case 100)

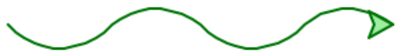
b) Use the **`draw_arc`** function to define a new function

```
def draw_squiggle(t, angle, num_segments, length, num_squiggles)
```

where each squiggle is two arcs of the given **`length`** with opposite angles. For example, a call to

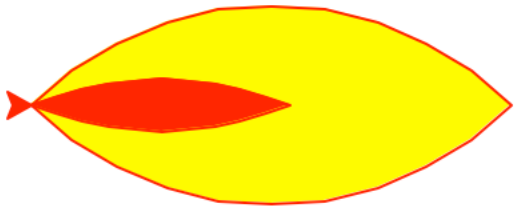
```
draw_squiggle(alex, 90, 10, 40, 2)
```

results in the figure below:



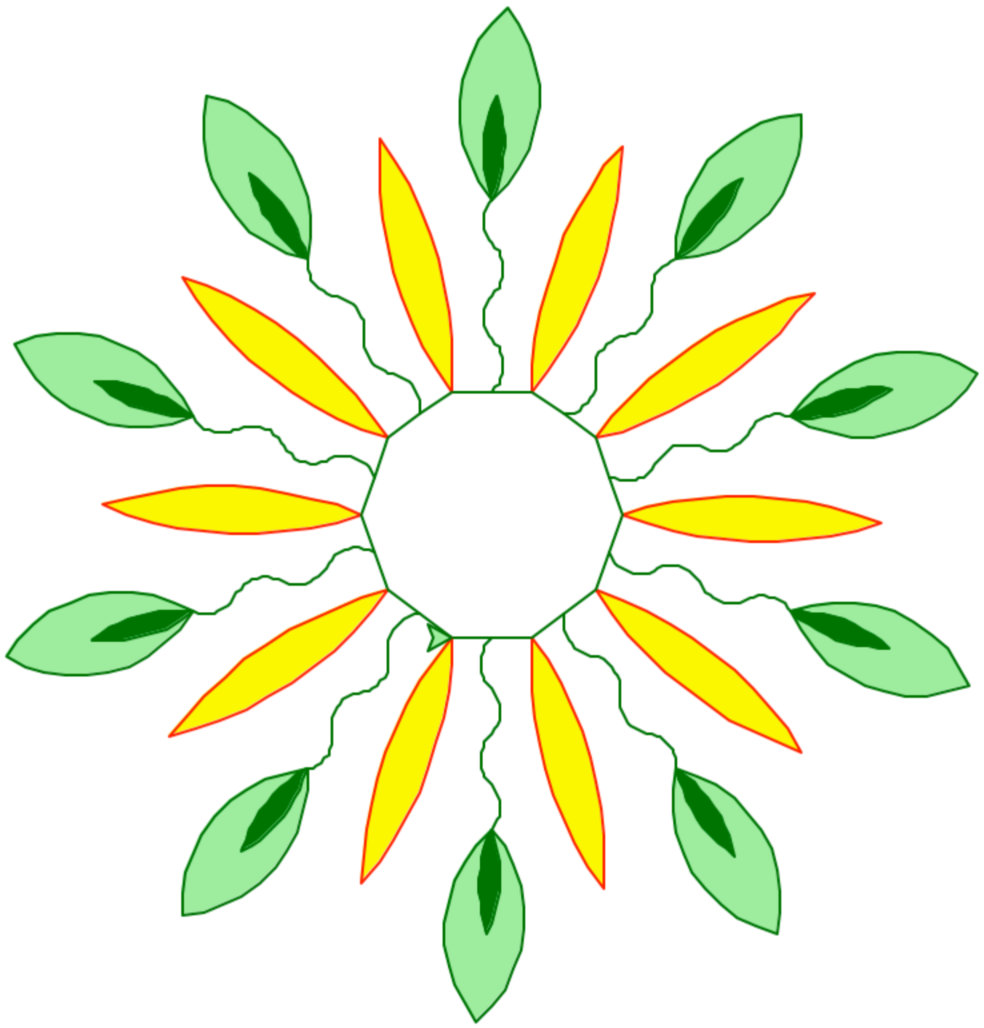
(Note: to draw the second arc that goes the opposite direction, you don't need to rewrite **`draw_arc`** - just call it with a negative angle!)

c) Use the function **`draw_petal`** to make a new function **`draw_leaf`** that has the same parameters, but makes a second petal inside the first one. The second petal has half the length and half the angle of the first one. Its fill color should match the pen color. For example, if the turtle **`alex`** starts out with pen color red and fill color yellow, **`draw_leaf(alex, 90, 10, 200)`** would give us this:



To change the fill color into the pen color for a turtle **`t`**, just use the turtle function **`pencolor()`** that returns the pen color, e.g., call **`t.fillcolor(t.pencolor())`**

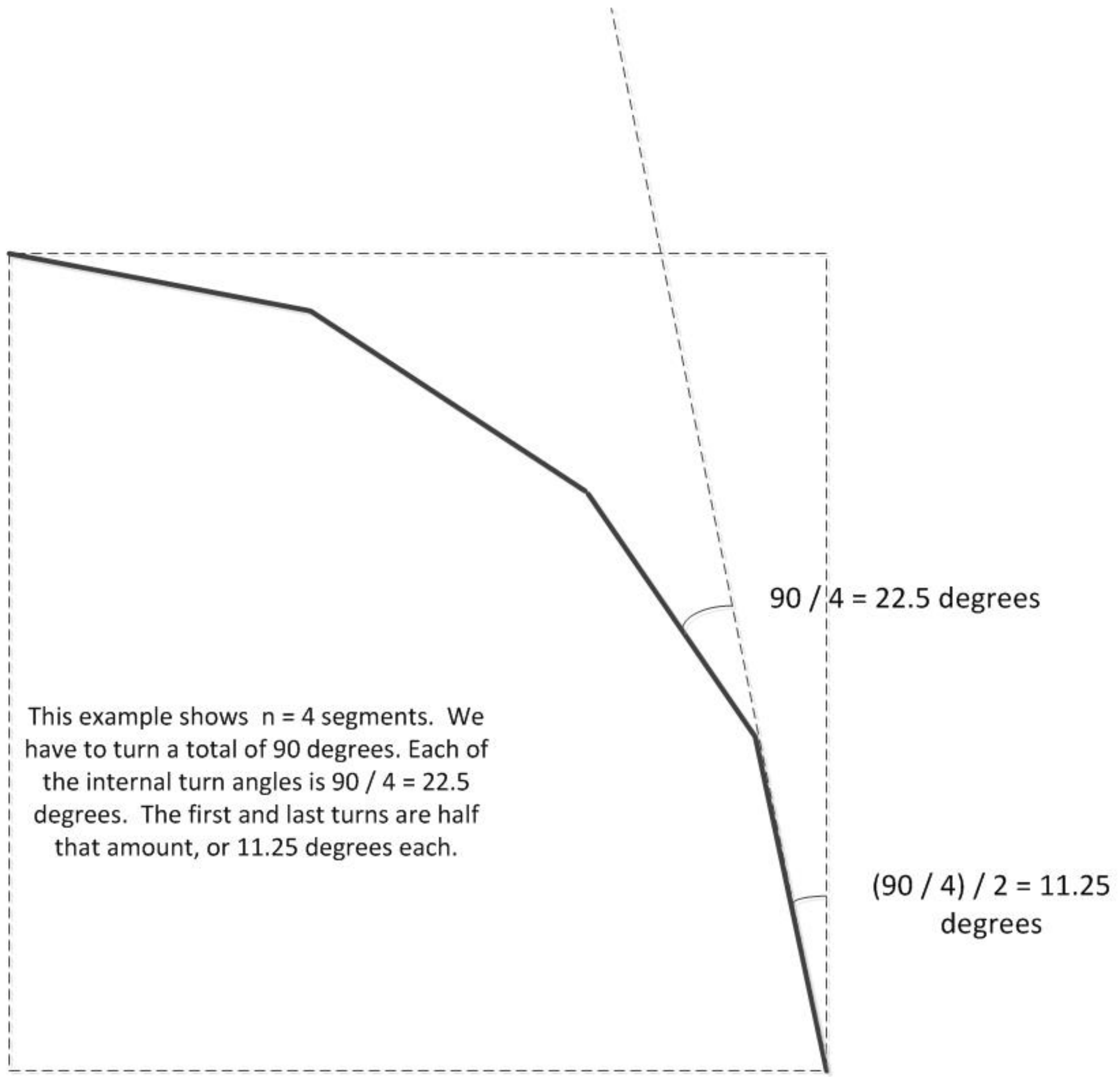
d) Compose the functions above into a program that draws a figure like the one below. Use any combination of four colors that you want; this one uses "red", "yellow", "light green" and "dark green". You can find a list of all turtle colors here:
https://www.tcl.tk/man/tcl8.4/TkCmd/colors.htm



Problem 2

a) Create a function **`draw_square`** that draws a square of given size (with a given turtle, of course, as in the functions above). The turtle should end up at the same point and facing the same direction as it started.

b) Create a function **`draw_quarter_circle`** that draws a quarter circle that fits within a square of given size. This is a slightly different problem than the one we solved for drawing an arc for the petal of the flower, because here we have a fixed point where we have to end up, namely, the opposite corner of the square. The turtle should end up at the opposite corner of the square, having turned a total of 90 degrees. It is easiest to see the problem with a small number of segments in the arc. Here is a sketch using four segments (assuming the turtle starts at lower right and is headed "north" (up the page)).


90 / 4 = 22.5 degrees
This example shows n = 4 segments. We have to turn a total of 90 degrees. Each of the internal turn angles is 90 / 4 = 22.5 degrees. The first and last turns are half that amount, or 11.25 degrees each.
(90 / 4) / 2 = 11.25 degrees

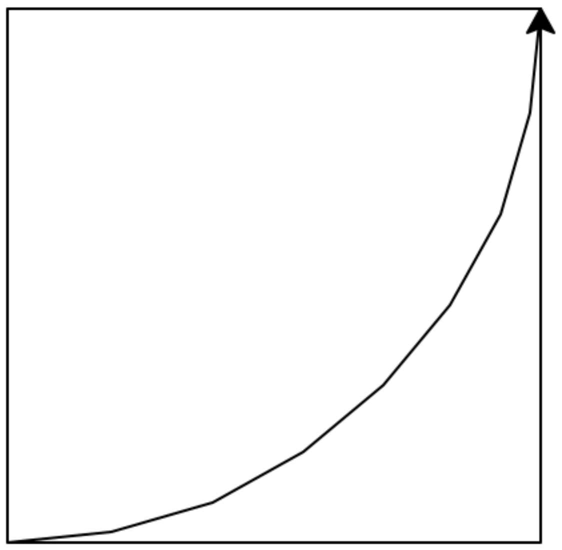
In general for *n* segments each internal turn angle is just 90/*n* and the first and last angles are half that amount (making a total of 90 degrees). The tricky part is that since the size of the square is fixed, we need to calculate the correct length for each segment. This requires a bit of trigonometry, and you can just use the calculation:

```
    segment_length = 2 * size * math.sin(math.radians(turn_angle / 2))
```

The parameters for the function should be the turtle, the size of the square, and the number of segments. As a test, if you have a turtle named **`alex`**, you should now be able to call

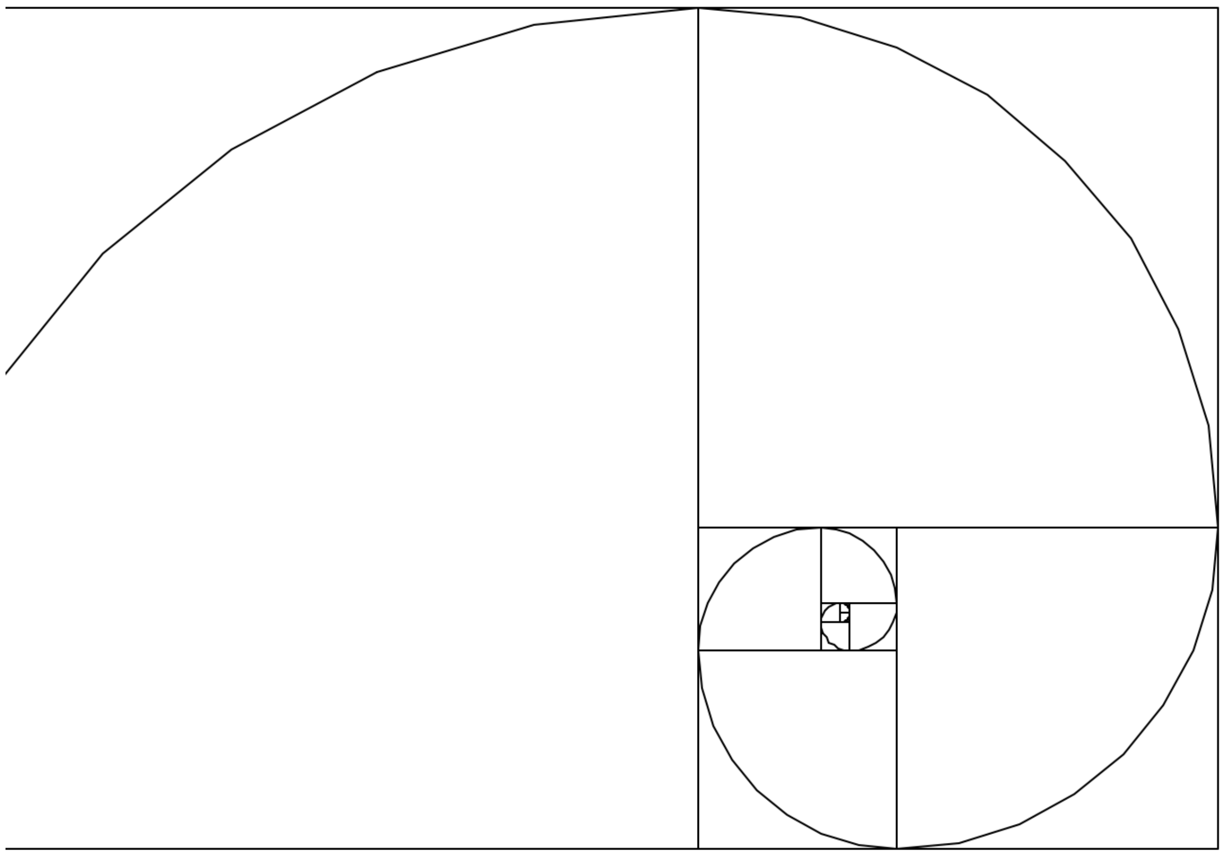
```
draw_square(alex, 200)
draw_quarter_circle(alex, 200, 8)
```

and see the following:



c) Now we can compose the functions above into a program that creates a lovely figure known as the *Fibonacci spiral*. See the screenshot on the next page, showing 10 iterations with an initial square of size 5. The size of each larger square is just the size of the current square plus the size of the previous square. Drawing the quarter circle automatically puts the turtle in position to draw the next square. We need an extra variable to keep track of the size of the previous square. In pseudocode, you're just doing this:

```
previous = 0
current = initial size you choose
for however many iterations you want
    draw a square with current size
    draw a quarter circle with current size
    new_size = previous + current
    previous = current
    current = new size
```



If you have questions

For questions, please see the Piazza Q & A pages and click on the folder `hw1`. If you don't find your question answered, then create a new post with your question. Try to state the question or topic clearly in the title of your post, and attach the tag `hw1`. *But remember, do not post any source code for the classes that are to be turned in.* It is fine to post source code for general Python examples that are not being turned in. (In the Piazza editor, use the button labeled "pre" to have code formatted the way you typed it.)

If you have a question that absolutely cannot be asked without showing part of your source code, make the post "private" so that only the instructors and TAs can see it.

Documentation and style

Include comments at the top of each file with

- your name, and
- a brief statement of what the program does

For each *function* you write, include a comment that states what it does. For functions, Python programmers prefer to put such comments in the form of a "docstring". A docstring is just a string of text, appearing directly after the function's def line, surrounded by triple quotes. See the sample code arc_experiment.py for examples.

What to turn in

Turn in two files, one for problem 1 called **`flower.py`** and one for problem 2 called **`spiral.py`**. Upload your two files to the Homework 2 submission link on Blackboard. Be sure to CHECK your submission history and make sure that you successfully submitted both files. **Remember you can't just upload the files, you have to click the "Submit" button!**