CS5001
Lab 5
October 3, 2019

Turtles, Dicts, and Recursion
Lab written by Chris Gregg and Mark Miller

Lab Part 1: Download the starter code

Download the starter code project from the following web page:

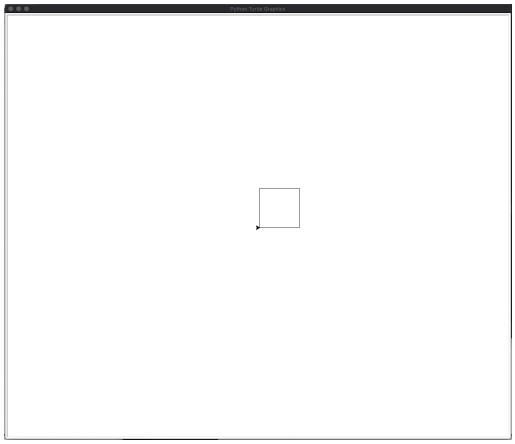
https://course.ccs.neu.edu/cs5001f19-sf/static/code/Lab5.zip

Open the project in PyCharm, and open the **turtle-fun.py** program inside the project.

Lab Part 2: Turtles!

In tonight's lab, you are going to be working with *turtle graphics*, which was a major component of the Logo programming language, designed to teach children how to program. A turtle graphics module is included with Python.

Go ahead and run the turtle-fun program in PyCharm. You should a new window pop up and a drawing of a square happening on the screen:



The triangle in the diagram is the *turtle*, and you give the turtle commands to draw on the screen.

To stop the program, click anywhere on the turtle screen.

Lab Part 3: The **`square`** function

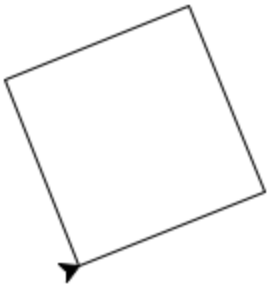
3a) In the code, take a look at the **`square`** function. The function call, **`turtle.forward(100)`** means "draw a line 100 pixels long" and **`turtle.left(90)`** means "turn the turtle left by 90 degrees." Notice that it is a verbose function that can easily be changed to use a **`for`** loop. Make that change now, and re-run. The function should now be much shorter.

3b) Next, notice that the function can only draw a single square, with a side length of 100. Change the function so that it has a single parameter, **`side_length`**, and then modify the function body so it uses the side length to draw the square. In the **`if __name__ == "__main__"`** block, add an input to the user to ask for the side length of the square, and pass that into the **`square`** function. Example run:

I am going to draw a square. How long would like the side length to be? 300
<then the turtle draws a 300-side length square)>

3c) Add another *optional* parameter called **`initial_rotation`** (with a default of 0) that lets the user specify the orientation of the turtle when it begins drawing. Add an input for the user to specify the initial rotation. For example:

I am going to draw a square. How long would like the side length to be? 100
What would you like the initial rotation to be? 22



Lab Part 4: Triangles, Pentagons, and Polygons

4a) There are two other functions, **`triangle()`** and **`pentagon()`**, which should draw triangles and pentagons (5-sided polygons). Write the body of the functions to behave like the **`square`** function from part 3b above. You will have to figure out the angles needed for the rotation.

4b) Now that you have the basic idea about how to draw some small polygons, go ahead and work on the **`polygon`** function, which requires a

side_length and a **num_sides**. You will have to do a tiny bit of arithmetic to make it work for any type of polygon.

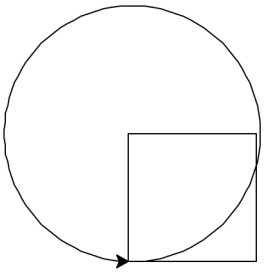
Lab Part 5: Circle

Now work on the **circle** function (note that there is a built-in circle function, but you will write your own approximation for this function). The **circle** function takes a radius. We have pre-defined a constant, **NUM_SIDES** to be 100. You should use the **radius** and **NUM_SIDES** to approximate a circle, using your **polygon** function. *Hint:* the circumference of a circle is calculated by $2\pi r$. How can you use the circumference to approximate how long each side should be?

To test your circle function, you can run the following:

```
circle(100)
square(100)
turtle.exitonclick()
```

and the drawing should look something like this:



Lab Part 6: Words to numbers

Add the following lines of code in your **if __name__ == "__main__"** block:

```
type_of_shape = input("What type of shape would you like to draw? (triangle, square, pentagon, sextagon, septagon, octagon, nonagon, decagon, dodecagon, circle) ")
side_length = int(input("How long should each side be? "))
```

What could you do to translate the user's answer to the first question to be useful to the **polygon** or **circle** functions? You could be thinking, "Oh, I could have a bunch of **if** statements, and then translate to side lengths that way..." But, that would be a lot of code.

Instead: create a **dict** that has the shape names as keys, and the shape side lengths as their respective values. This way, you can easily pass in the side length

to the **`polygon`** function. (Keep in mind that you will still need a special case for circle!).

Lab Part 7: Recursion!

In class, we discussed recursion, which is when a function calls itself. When writing a recursive function, you have to perform three steps:

1. Check the *base case* and stop if the base case condition is met.
2. Work towards the base case (by performing part of the problem)
3. Call the function recursively with a *smaller* problem of the same form

The first recursive function we'd like you to write will draw a *spiral*, which should look like this:

(the drawing here loops every 10 seconds or so -- you only have to draw it once)

Fill in the recursive function, **`draw_spiral(line_len)`**, to draw the spiral. Here are some hints:

1. First, draw the spiral on paper. What do you have to do at each step? E.g., first, draw a line that is **`line_len`** long. Then turn 90°. Then...
2. Once you figure out the steps so you understand the problem, start "thinking recursively":
    a. What is the base case? How short can a line be before it is not a line any more?
    b. Work towards the base case. What steps will make the next part of the spiral?
    c. Call the function recursively with a *smaller* problem of the same form. Go back and review your paper drawing -- what did you have to do for each line?
3. Test your function! Call **`draw_spiral(100)`** to see it go. If it has problems, brainstorm some more, and then ask for help if you still can't quite get it.

Finally: your polygon function could be converted to a recursive function. Think about what you would have to change to make it recursive. If you want to try it, create a new function called **`recursive_polygon()`** and write the function there.

Hint: you will need a parameter to keep track of how much farther you have to go in the polygon -- remember, there has to be a base case!

Lab Part 8 (Bonus Fun):

Here is a list of turtle methods:

http://opensask.ca/Python/MoreTurtles/LoopsReview.html#summary-of-turtle-methods

Add some flair to your functions above by adding more parameters for the user. Here are some ideas:

1. Have the user input a line color, or fill color.
2. Have the user ask for *n* number of polygons, randomly placed on the screen (hint: use the **`turtle.penup()`** and **`turtle.pendown()`** functions to lift up and put down the pen, and use the **`goto()`** function to jump to a random location).
3. Make a *spiral* of a particular polygon.
4. Make *concentric* polygons, that are centered on a particular location.

Lab Part 9:

Submit your lab at http://handins.ccs.neu.edu