Fall 2015 COM110: Lab 9  
Recursive drawings, fractals

1) Recursion can be used to make interesting drawings. In order to set the stage for drawing recursively, we will make use of something called Turtle graphics. Picture an invisible “turtle” who can only do three very specific things:

- travel forward a certain distance while leaving a trail (i.e. “drawing”),
- change direction (i.e. “turning”), and
- move to a specified point (as though someone picked the turtle up and placed it down at the specified point without changing the direction it’s facing or leaving a trail).

a) Open up turtle.py to see these commands realized as a Python class called Turtle. (Don’t worry about understanding the code in the body of the methods, but read all the method signatures and their documentation and comments.) A Drawing window with a Turtle object has already been created for you in the main function. Ask a TA or instructor any questions you have about what each method does!!
b) Note that until you draw() with the turtle, you will see nothing. (The turtle itself is not a graphical object, so you can think of it as “invisible.”) Try draw()-ing a line of length 100.

c) Try out a few more turtle commands to see how the invisible “turtle” draws and moves. Notice that in the main method, the Turtle object has already been moved to the left side of the graphical window, halfway up. Also note that when you turn the turtle, the degrees of rotation must be given in radians. So to turn the turtle 90 degrees, you must turn by \(\frac{\pi}{2}\) radians (since \(\pi\) is 180 degrees). And to turn 60 degrees, you must turn by \(\frac{\pi}{3}\) radians. Positive radians means turn the turtle to the left (counterclockwise). So if you want to turn the turtle right (clockwise) you will have to put a negative sign in front of your radians. E.g., To turn the turtle 60 degrees to the right you must turn by \(-\frac{\pi}{3}\).
d) Practice using the turtle commands (turn(), moveTo(), and draw()) by drawing your initials in the graphical window. (It is a good idea to sketch out your route for the turtle on a piece of paper before diving into this.)

2) Now let’s create a drawing using recursion. Recall that any recursive function always needs a base case under which the recursion stops (i.e., no more recursive calls are made). You might think of each recursive call as another “level” of recursion. (Think of the recursive call stack and how its depth grows with each recursive call.) If each time we make a recursive call, we imagine going “down” one “level,” we can think of our base case as being when the “level” reaches 0, and this is when we stop making recursive calls. Keeping this in mind...

a) Let’s create a square spiral by drawing a certain length in one direction, turning left 90 degrees and then calling the function recursively with a smaller length and with “level” reduced by one. Complete the turtle.py spiral() method by filling in the code for the recursive case. In the recursive case:

i) draw for the given length,

ii) then turn 90 degrees to the left,
iii) then recursively call a “smaller” instance of \texttt{spiral}(): call it with \textit{reduced} parameter values... specifically, call \texttt{spiral} with a \textit{reduced} length (scaled down by, say, 14/15 from the original length) and a \textit{reduced} level (level - 1).

b) Test your \texttt{spiral}() function by calling it from the \texttt{main} function.

\textbf{Get check 3}\textbf{\textcolor{red}{}}

3) \textbf{Fractals} are geometric objects that exhibit \textit{self-similarity} and “infinite complexity” as you “zoom-in” on them closer and closer. It was not possible to generate even partial pictures of fractals until we could harness the power and speed of computers. The Mandelbrot set is one example of a fractal. See the series of images on the right hand side of this page: http://en.wikipedia.org/wiki/Fractal.

4) The above link shows a series of images of the same picture at progressively higher zoom levels. You can see from this sequence of images that as you zoom in to the center point of the picture further and further, you eventually \textit{again} find the original Mandelbrot set! Also see this video.

5) Just as we’ve learned that a recursive function keeps invoking smaller and smaller instances of itself, fractals also exhibit this kind of repeated and self-similar behavior. Indeed, using recursion is a common way to generate images of fractals. Take a look further down the \texttt{wikipedia} page on fractals to see other examples of famous fractals. Look especially at the animation/description of the Koch snowflake. Here is another page about the Koch snowflake: http://mathworld.wolfram.com/KochSnowflake.html.

6) You can also learn about the Koch snowflake by reading exercise 8 on page 464 of Zelle. Creating this snowflake will be our next task.

a) First, call \texttt{kcurve} from your \texttt{main} function with \texttt{level 0}. It should draw a straight line. Up the \texttt{level} to 1 for the next step.

\textbf{Get check 4}\textbf{\textcolor{red}{}}

b) Now we must complete the recursive case of the \texttt{kcurve()} function. Imagine the turtle standing on one corner of the triangle, facing the next corner in the clockwise direction. (The picture in your text will help with the visualization here, or you can sketch one yourself.) To draw a Koch curve between these two corners, take the following steps.

i) You need to reduce \texttt{length} to 1/3 of the old length and reduce the \texttt{level} by 1.

ii) Call \texttt{kcurve()} with this new length and reduced level.

iii) By doing this you’ve reached 1/3 of the way across the side of the triangle so it’s time to form that “equilateral bump” that \texttt{wikipedia} and our book talked about.

iv) How many degrees are in each inner angle of an equilateral triangle? Turn to the left that many degrees and then draw your new spur (by calling \texttt{kcurve()} again).

v) You are now out on the tip of your “equilateral bump” so you must make your way back to the side of the triangle. Turn right 120 degrees (the complement of the inner angle of the triangle), and create a spur going back (by calling \texttt{kcurve()} again).

vi) Finally, \texttt{turn left} to finish the final third of the side of the triangle, again calling \texttt{kcurve()} so that you arrive at the destination corner!
c) Once you’ve done all this, when you increase the level of recursion in your calls to \texttt{kcurve()} in the main function, the top of your Koch snowflake should appear! (Try recursion levels of 2 and 3.)  \textbf{Get check 5}


d) As you can see from the above websites, the Koch snowflake is, at the start, just a triangle. To complete the snowflake, in the \texttt{main} function, create an equilateral triangle of \texttt{kcurve()}’s. i.e., after the \texttt{kcurve()} you’ve already called in your main function, turn right 120 degrees (the exterior angle of an equilateral triangle), call \texttt{kcurve()} again, turn once more, and call \texttt{kcurve()} for the final time to make the third side of your triangle. (Hint: In radians, 120 degrees is 2*pi/3.) Your snowflake should be complete!

\textbf{Get check 6}


7) Koch Snowflake follow up:
   a) Try setting each side of the triangle to a different level of recursion.
   b) Try changing all right turns to left turns and vice versa. What happens to your snowflake? Can you explain what’s going on?

\textbf{Get check 7}


Bonuses

As always, email your bonuses to Professor Chung after getting them checked by a TA!

\textbf{Get these bonus checks in any order one at a time}

A. Complete programming exercise 4 on page 463 of Zelle. (Write the max() function recursively.)
B. Complete programming exercise 9 on page 466 of Zelle. (More fractal fun!)

The following two (non-recursive) exercises are from \textit{Python Programming in Context} by Miller and Ranum

C. Research the UPC codes that are used on products of all kinds. Use the turtle module to draw a UPC code for a given product name and price.
D. Research the US postal bar codes used to encode a zip code. Use the turtle module to draw a postal code for a given zip code.
E. (Simulation and Randomness.) Read first four paragraphs here about the Monty Hall problem: \url{http://en.wikipedia.org/wiki/Monty_Hall_problem}. Create a simulation that demonstrates switching your door choice after the host opens a door without the car (surprisingly) \textit{does} improve your odds of winning the car.