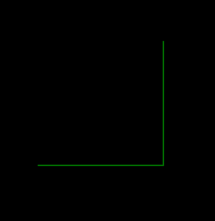
**Dragon Curve Fractal using Loops**

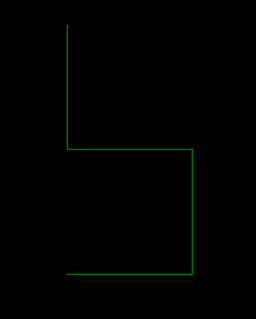
We start off with 2 python lists called “new” and “old”. Both of them are character lists. The elements of “new” define the order in which our turtle will have to move in each iteration. On the other hand, “old” stores the movements of the previous iteration. I have also used a third list called “steps” which stores the movements across all the iterations combined. I created this list so that we can simply loop through all the elements of “steps” and move our turtle accordingly. Each of the elements in our lists can be ‘f’, ‘r’ or ‘l’ (which stand for forward, right and left respectively). Iterations of the Dragon curve can easily be generated by folding a strip of paper n number of times. Take a strip of paper and fold it in half to the right. Fold it in half again to the right. Just continue this process as many times as you want (of course it will be hard to fold after a certain point). Let’s say you do it 2 times.

Now unfold the strip and relax it, and you’ll notice that every bend/corner of the strip has become a 90 degree turn. What you have now is the second iteration of the Dragon Curve. If you fold the strip of paper in half for a third time and then open it up, you would get the third iteration and so on.

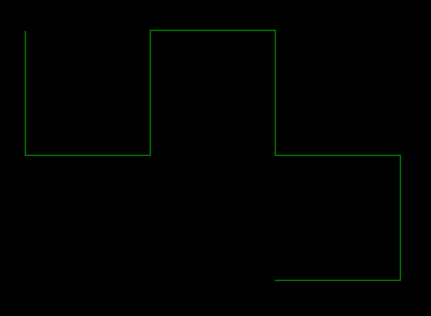
First Iteration-



Second Iteration-



Third Iteration-



And so on…

Note: In python, by default, the turtle faces towards right direction.

Now all you have to do is visualize the turtle walking along these lines and it should be able to predict how to move.

1st iteration- F L F L

2nd Iteration- F L F L F R F L

3rd Iteration- F L F L F R F L F L F R F R F L

If you know the n th iteration, you can predict the (n+1) th iteration. Here’s how…

Let’s say we are trying to derive the third iteration from the second iteration.

1. Ignore the last element for now. (F L F L F R F)
2. In the remaining elements, if there’s an R, replace it by L. And if there is an L, replace it by an R. (F R F R F L F R)
3. Now flip these elements about the mid point. (R F L F R F R F)
4. Add the last element which we had ignored in the first step. (R F L F R F R F L)
5. Append the result of step 4 to the 1st iteration and you will have your third iteration ready!

Using this same logic, you can predict the next iterations.

So like I said earlier, I used “old” list to keep track of the previous iteration and “new” list to define the current iteration. To complete the algorithm, replace all the elements of “old” with “new” and repeat the above steps for the next iteration.

Input- The user has to mention the number of iterations and the size. I would recommend you to input iterations less than *20* or less, otherwise the program may crash.