Sierpinski Triangles

October 31, 2019

1 Homework 8 - Q4

To begin with, I've put a little playable number just below this cell, called depth, which will enable anyone using this Jupyter Notebook to change the depth to which the Sierpinski Triangles are rendered.

To meet the exact requirements of this question, I have used `depth = 4`. Feel free to try other variants of this code, which you can get from my GitHub.

[1]:
```
depth = 4
```

Our first step is to create a triangle boundary inside which we will plot all the other triangles.

We're doing that with 3 points, and since we want to later plot over the same plot, we're keeping it inside an list, p, to which we will keep adding all our other plots later on too.

Also, since we don't want the triangle to be filled, we're using the

[2]:
```
A = [0,0]
B = [10/2,10]
C = [10,0]
p=[]
p.append(polygon([A,B,C],fill=False))
```

Now, we need to find a lot of mid points, so might as well create a function `mid(A,B)` to find the mid point of the points `A` and `B`:

[3]:
```
def mid(A,B):
    return [(A[0]+B[0])/2,(A[1]+B[1])/2]
```

Now, we want to plot another triangle with the mid points of the previous one. So, we do the following:

[4]:
```
p.append(polygon([mid(A,B),mid(B,C),mid(C,A)],fill=False))
```

Now, the actual fun starts... (Or at least that's what I feel, cuz since I had to devise this one myself, it felt a lot harder to me than it probably looks to you)

We're basically going to create a list `todolist` in which we will store the points of the triangle inside which we want to draw triangles in the next couple of iterations.

Next, we're creating a function called `gothrough`, which does precisely that... It goes through and plots triangles inside the supplied triangle's coordinates.

The coordinates given as a 3×2 list of coordinates, like this:

$[A_x \quad A_y]$

$[B_x \quad B_y]$

$[C_x \quad C_y]$

Now, we have to also add these to the same plot that we were plotting to earlier, for which we use the next parameter, `plots`, into which we append the new plots we make.

Now, the logic for the plotting is simple:

Find the mid-points of the triangle, then plot the sub-sub-triangles formed by joining the mid points of the lower-left sub-triangle. This is followed by the same for the upper sub-triangle and then the lower-right sub-triangle.

Basically, we're moving from one sub-triangle to another sub-triangle and plotting the sub-sub triangles in each of them, while at the same time, storing the locations of the sub-triangles in our `todolist` so that we can perform the same task in the sub-triangles later...

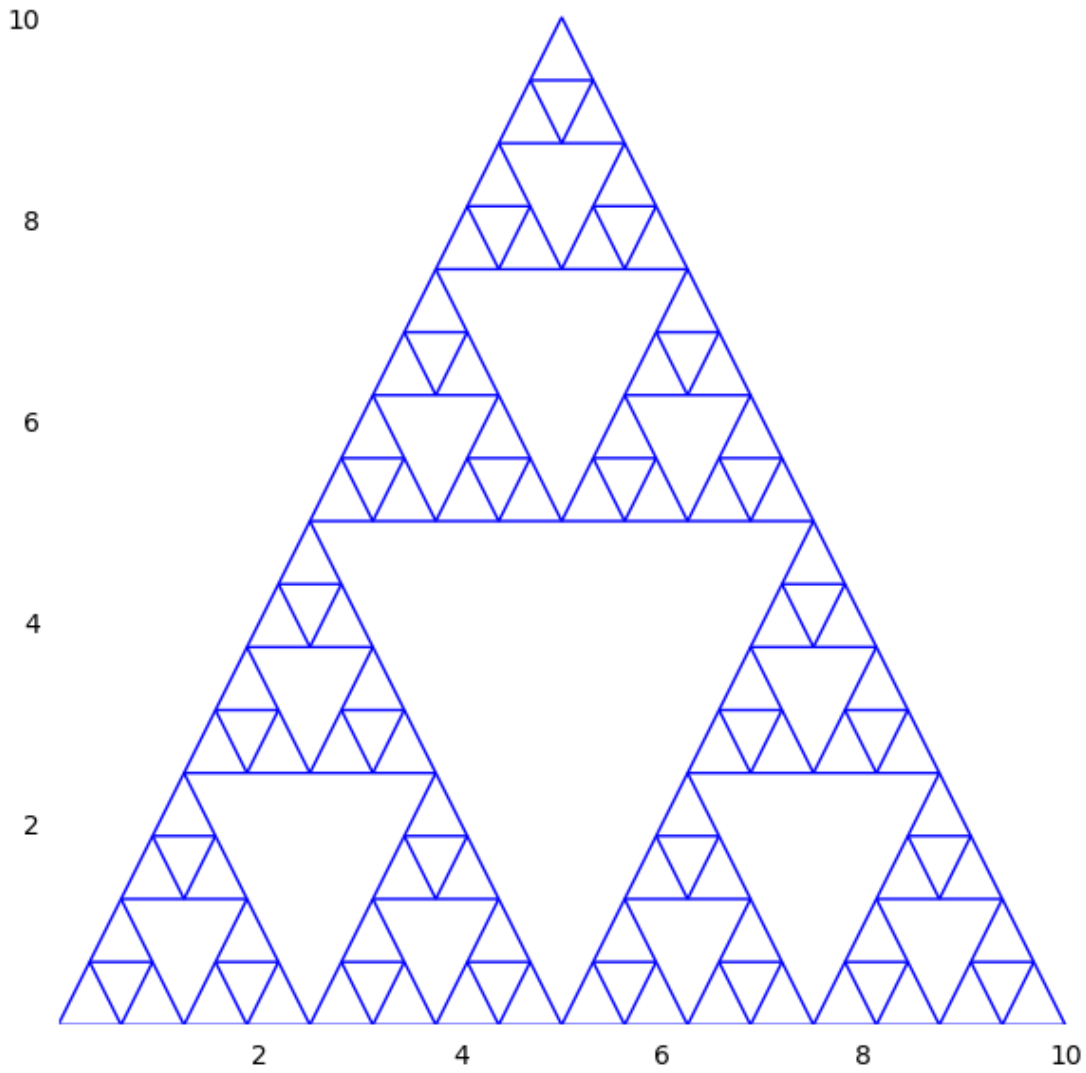
Go through the code and try to figure out what is happening in case this logic looks weird or is not explanatory enough.

[5]:
```
todolist=[]
def gothrough(P,plots):
    A = P[0]
    B = P[1]
    C = P[2]
    D = mid(A,B)
    E = mid(C,B)
    F = mid(A,C)
    plots.append(polygon([mid(A,D),mid(F,D),mid(A,F)],fill=False))
    todolist.append([A,D,F])
    plots.append(polygon([mid(B,D),mid(B,E),mid(D,E)],fill=False))
    todolist.append([D,B,E])
    plots.append(polygon([mid(F,E),mid(C,E),mid(C,F)],fill=False))
    todolist.append([F,E,C])
```

Now, we're trying to figure out what happens when we choose a depth less than or equal to 2, since this function is generalised only when the depth is more than 2. In the other cases, we have to hard code it, which is fairly alright, since, recursive functions generally have the lowest one or two cases defined from before (e.g. Fibbonacci Numbers and Factorial...)

[6]:
```
tot=0
for i in range(depth):
    tot+=3**i
if depth==1:
    t =sum(p)
```

```
    t.axes_color('white')
    t.show()
elif depth==2:
    gothrough([A,B,C],p)
    t=sum(p)
    t.axes_color('white')
    t.show()
elif depth>2:
    gothrough([A,B,C],p)
    for i in range((tot-4)/3):
        gothrough(todolist[i],p)
    t = sum(p)
    t.axes_color('white')
    show(t, figsize=8)
print("Behold the Sierpinski Triangle Fractal!\n(till depth",depth,")")
```



```
Behold the Sierpinski Triangle Fractal!
(till depth 4 )
```