Introduction to Programming - Project Part 1

Name: Danny Rene Jensen
Supervisor: Peter Schneider-Kamp
DM536

30. oktober 2014

Indhold

1 Specification

1.1 Sirpinski triangle

The program should be able to draw a sierpinski triangle of x depth and y length, by the help of TurtleWorld. I am using FractalWorld, which is a more specific version of TurtleWorld.

1.2 Fractal tree

The program should be able to draw a fractal tree of x depth and y branch length, by the help of TurtleWorld. I am using Fractal World, which is a more specific version of TurtleWorld.

1.3 Class program

This program should be able to take any .fdl file and be able to draw it's respective drawing, with the depth, length, scales and rules, that is defined in the .fdl file.

2 Design

2.1 Sirpinski triangle

First the program needs to create a world and then a turtle. After this, the turtle needs to be positioned at a specific place. When that is done the turtle will start to draw the first three small triangles and then move onto the next three and so on, until it has drawn all the triangles for the specific depth.

2.2 Fractal tree

To begin with the program needs to create a world and then a turtle. Then the turtle needs to be positioned at a specific place. When that is done the turtle will start to draw the tree trunk. After that it will go all the way out to the right and then back up and turn left and go out, until the whole tree is drawn.

2.3 Class program

The program should read the .fdl file first, then it shall be able to save the separate inputs ie. depth, length and the other from the fdl file. Next it needs to be able to take the commands and see how it needs to run them. After it is done, it should draw the respective drawing.

3 Implementation

```
her kan i vise kode som i forklarer.
        indryk virker fint her.
```

3.1 Sirpinski triangle

```
def triangle(t, l, depth):
 if depth <= 0:
  #here the turtle draws the triangles
  for i in range(3):
   fd(t, l)
   lt(t, 120)
```

This program needs to make a triangle, when the depth is 0. This is done in the cutout of the program above. If the depth is higher then 0, the program also needs to go to the next triangles place and then execute depth 0. This happens with a recursive call in the else statement.

```
 else:
  for i in range(3):
   #here the turtle postions itself for the next triangle
   triangle(t, l, depth-1)
   fd(t, l*2**depth, depth-1) #depth here is the thickness of drawline
   lt(t, 120)
```

3.2 Fractal tree

```
 1  def tree(t,l,a,depth):
 2   if depth==0:
 3    return
 4   #experementing with colors
 5   if depth<4:
 6    set_pen_color(t,"#1a9108")
 7   elif depth==4:
 8    set_pen_color(t,"#d14500")
 9   elif depth==5:
10    set_pen_color(t,"#b23b00")
11   elif depth==6:
12    set_pen_color(t,"#993401")
13   else:
14    set_pen_color(t,"#7A2900")
15   pd(t)
16   fd(t,l/0.666**depth,2*depth-1)
17   rt(t,a)
18   tree(t,l,a,depth-1)
19   lt(t, 2*a)
20   tree(t,l,a,depth-1)
21   rt(t,a)
22   pu(t)
23   bk(t,l/0.666**depth)
```

First of all, the turtle needs to go all the way out to the right. But it needs to scale the length for every branch it draws. This scale happens in line 16 in the code above. It uses recursive to go all the way out, which happens in line 18. After that it needs to go back, so it can draw another branch to the left of the other branch. This happens in line 19, where it turns to the left and then once again uses recursive, line 20, to go out a branch. When it backs up it does not change the color, that is the reason why, in line 22, it picks up the pen right before it backs up, and puts it back down in line 15. If it did not do this the bright green color, from the mallest branch, would go all the way back in a thin line.

3.3 Class program

```
her kan i vise kode som i forklarer.
       indryk virker fint her.
```

```
if sys.argv[1]=='tree.fdl':
  pu(t)
  lt(t,90)
  bk(t,200)
  pd(t)
```

In this program the first if statement, that is not in a class is just to turn and place only the tree, so it grows up and is in the middle of the screen. After that the rule class starts.

```
1  class rule(object):
2   def __init__(self):
3    self.rule={}
4
5   def rule1(self,left,right):
6    self.rule[left]=right
7
8   def udv(self,lista,depth):
9    if depth<=0:
10    return lista
11   liste=[]
12   for i in lista:
13    if i not in self.rule:
14     liste.append(i)
15    else:
16     liste.append(self.udv(self.rule.get(i),depth-1))
17   return liste
```

In __init__ line 2 I define self.rule as a dictionary. To begin with I did not have an __init__ and the line self.rule= stood under my function rule1, line 5 in code above. This worked with all .fdl files, except dragon.fdl. The next function, udv, line 8, expands the right side of the .fdl rule list, it is doing this the depths times with recursive, line 16. If depth is 0, then it returns the start line from the .fdl, line 9 and 10, which is not always the same as the right rule.

```
class command(object):
 def __init__(self,cmd,a):
  self.cmd=cmd
  self.a=a

 def udf(self,turtle,length):
  if self.cmd=="fd":
   fd(turtle,length[0])
  if self.cmd=="bk":
   bk(turtle,length[0])
  if self.cmd=="lt":
   lt(turtle,int(self.a[0]))
  if self.cmd=="rt":
   rt(turtle,int(self.a[0]))
  if self.cmd=="scale":
   length[0]=length[0]*float(self.a[0])

class fractal(object):
 def __init__(self,start,regler,commando,length,depth):
  self.start=start
  self.regler=regler
  self.commando=commando
  self.length=[length]
  self.depth=depth

 def tegn(self,lista):
  for i in lista:
   if type(i)==list:
    self.tegn(i)
   else:
    cmd=self.commando.get(i)
    cmd.udf(t,self.length)

def read():
  commando={}
  r=rule()
  filex = open(sys.argv[1])
  for line in filex:
   line=line.strip()
   lista=line.split(' ')
   if 'start' in line:
    start=lista[1:]
   elif 'length' in line:
    length=int(lista[1])
   elif 'rule' in line:
    r.rule1(lista[1],lista[3:])
   elif 'depth' in line:
    depth=int(lista[1])
   elif 'cmd' in line:
    cmd=command(lista[2],lista[3:])
    commando[lista[1]]=cmd
  return fractal(start,r,commando,length,depth)
```

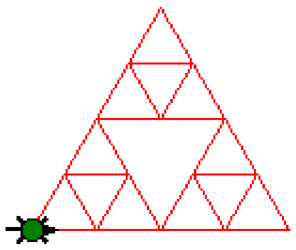
The command class is where the program takes the given letter from the .fdl file and turns it into something that the turtle can understand ei, F becomes fd(turtle,length). The length is made into a list, line 8, 10 and 16, because of the scale function. If it was not made into a list, the scale function would try to change the global length which it can't. Also the angle needs to be integers, that is why int(self.a[0])). The nop function was not programed, because it does nothing and if it does nothing it is as good as saying that it has no code. I could program it in with something like rt(turtle,360), but it would slow the program down and just be unrevelant fill code.

```python
class fractal(object):
 def __init__(self,start,regler,commando,length,depth):
  self.start=start
  self.regler=regler
  self.commando=commando
  self.length=[length]
  self.depth=depth

 def tegn(self,lista):
  for i in lista:
   if type(i)==list:
    self.tegn(i)
   else:
    cmd=self.commando.get(i)
    cmd.udf(t,self.length)

def read():
  commando={}
  r=rule()
  filex = open(sys.argv[1])
  for line in filex:
   line=line.strip()
   lista=line.split(' ')
   if 'start' in line:
    start=lista[1:]
   elif 'length' in line:
    length=int(lista[1])
   elif 'rule' in line:
    r.rule1(lista[1],lista[3:])
   elif 'depth' in line:
    depth=int(lista[1])
   elif 'cmd' in line:
    cmd=command(lista[2],lista[3:])
    commando[lista[1]]=cmd
  return fractal(start,r,commando,length,depth)
```

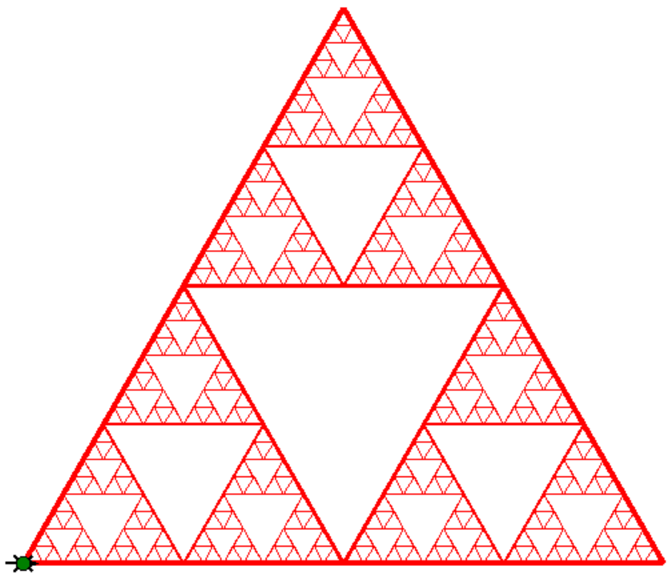
The read function is the one, that reads the .fdl file with the help of sys.argv, and gives all the names for the other functions. It also stores data in lista and asignes it to the names. It then tosses all the data over to the fractal class. This class is the one, that binds it all together. Last but not least the tegn function draws all of the inputs it gets.

4 Testing

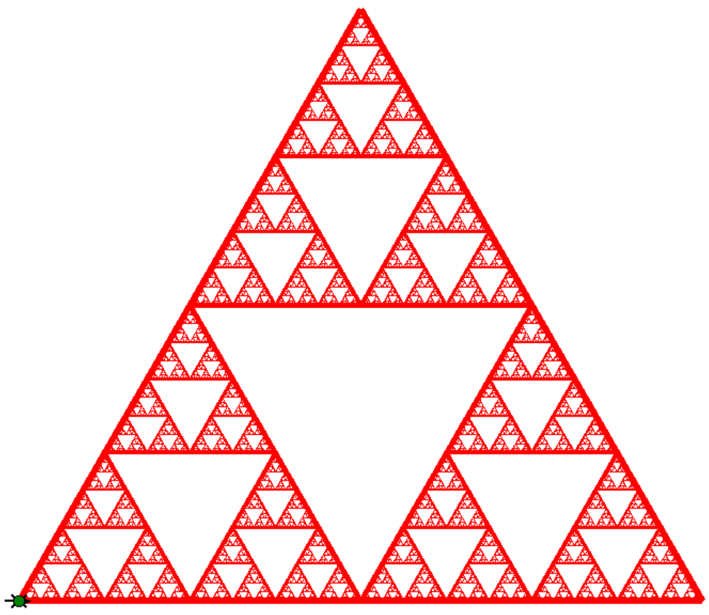
4.1 Sirpinski triangle



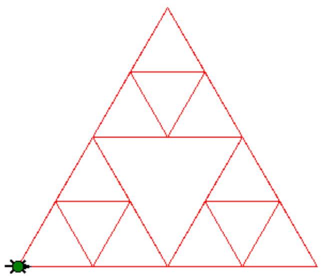
This is a sirpinski triangle with depth 2 and length 30.



This is a sirpinski triangle with depth 5 and length 15.

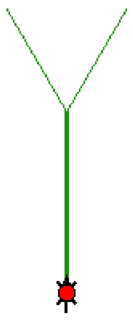


This is a sirpinski triangle with depth 7 and length 5.
Now, I want to test the program with 8/3 depth to see, if it can run with decimal numbers too:

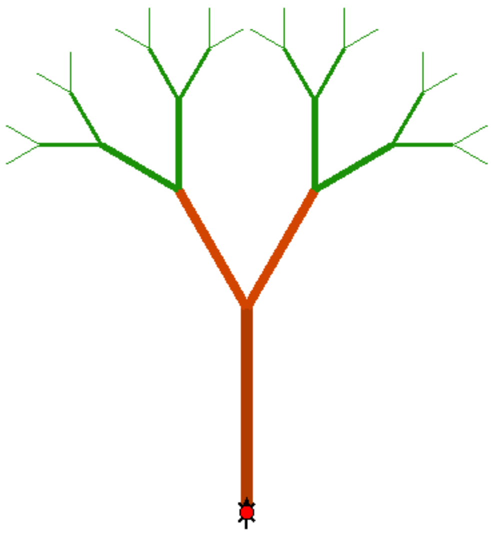


And suprisingly enough it does.

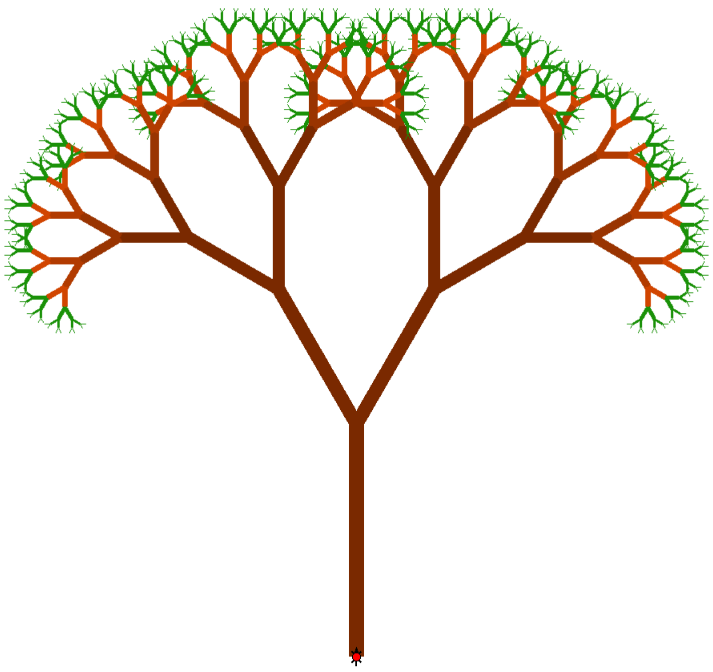
4.2 Fractal tree



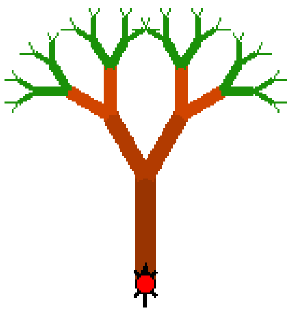
This is a fractal tree with depth 2 and length 50.



This is a fractal tree with depth 5 and length 20.

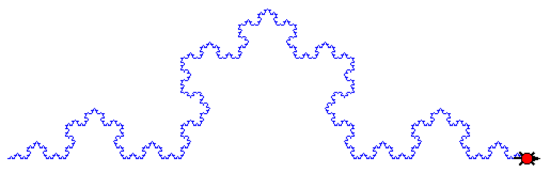


This is a fractal tree with depth 10 and length 5.
Can this program also handle decimal numbers?

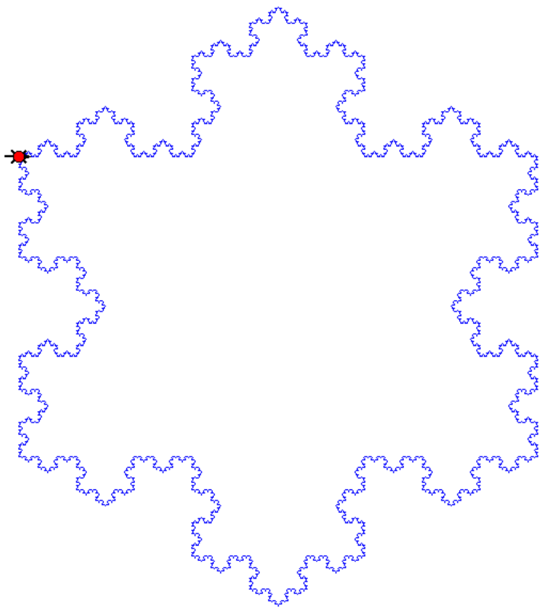


Yes, it can too. it draws the rounded down depth number too.

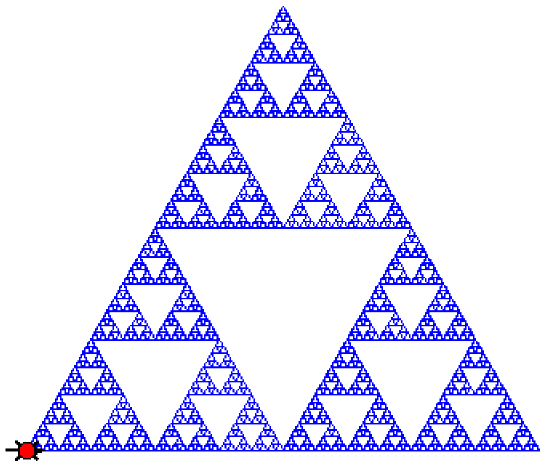
4.3 Class program



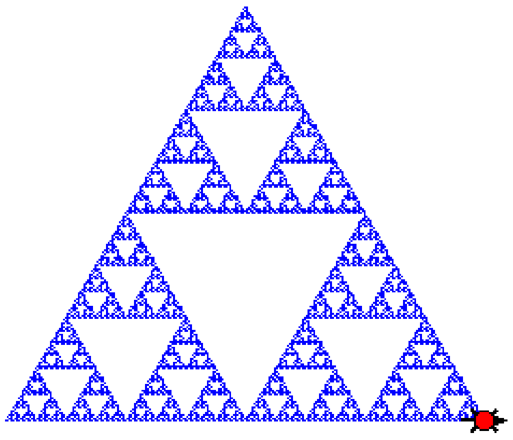
This is the koch.fdl with a depth of 5 and a length of 2.



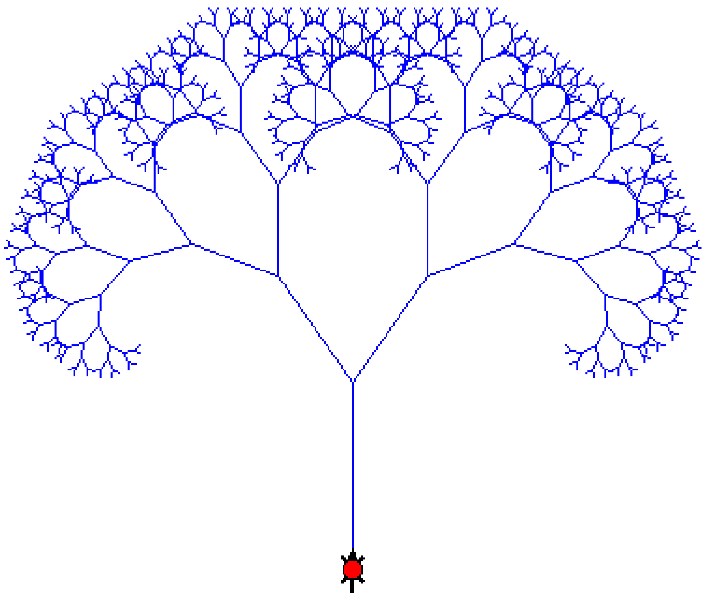
This is the snowflaske.fdl with a depth of 5 and a length of 2.(I assume the name should have been snowflake.fdl)



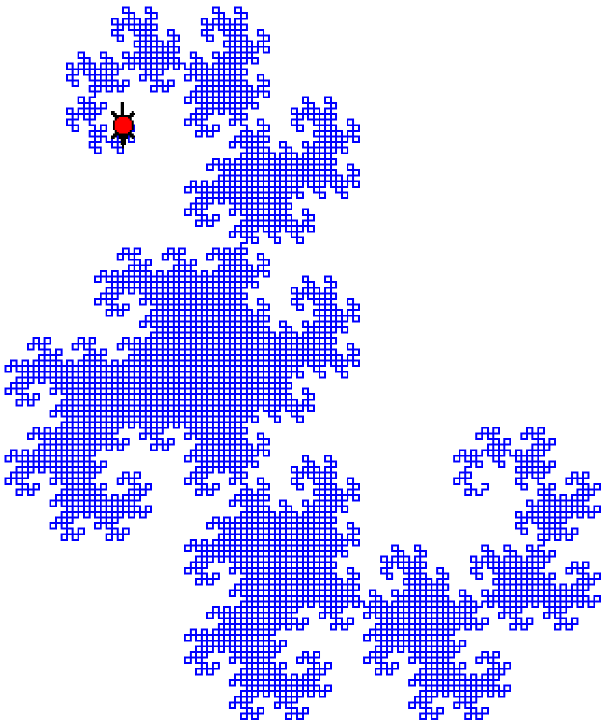
This is the siepinski2.fdl with a depth of 6 and a length of 5.



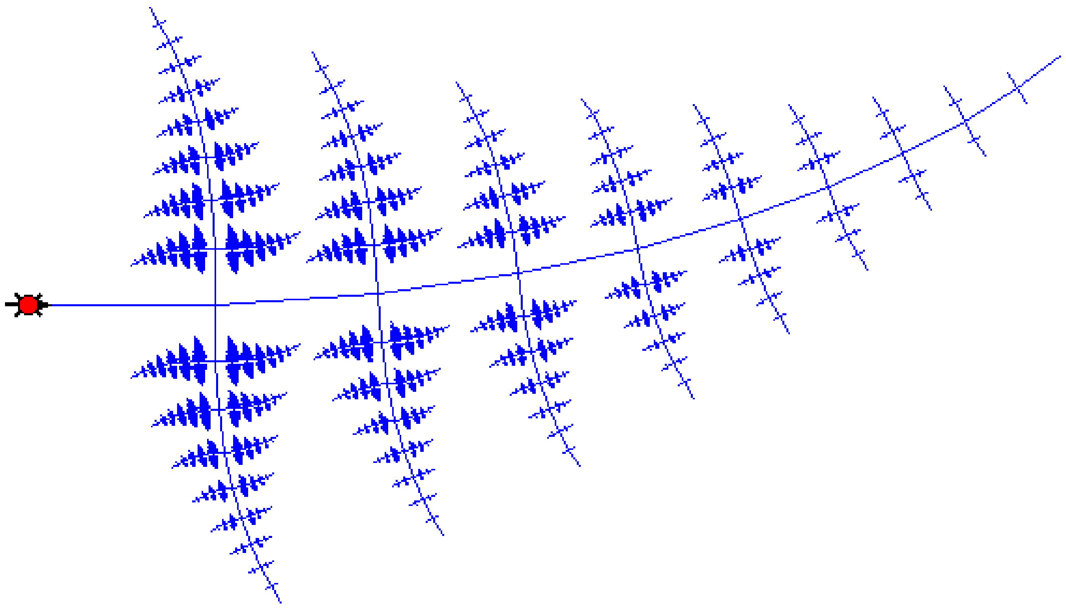
This is the sierpinski.fdl with a depth of 8 and a length of 1.



This is the tree.fdl with a depth of 10 and a length of 100.



This is the dragon.fdl with a depth of 13 and a length of 3.



This is the fern.fdl with a depth of 10 and a length of 100.(the depth 12, which the fern.fdl had at the beginning would simpely take to long time to draw.)

5 Conclusion

5.1 Sirpinski triangle

This program meets the requirements, set both by the assignment description and also mine, which is that I wanted another color for the drawing and the turtle. I also wanted different lines thickness for the different depths.

5.2 Fractal tree

This program also meets the requirements, set both by the assignment description and mine, which is that I wanted other colors for the drawing for the different depths. I also wanted different line thickness for the different depths.

5.3 Class program

The class oriented program works with all the .fdl it gets. It also turns the tree around, so it looks better.

6 Appendix (source code)

6.1 Sirpinski triangle

```python
from FractalWorld import *
world = FractalWorld(width=900,height=800)
turtle = Fractal(draw=True)
print turtle
set_pen_color(turtle, "Red")
set_color(turtle, "Green")

def triangle(t, l, depth):
 if depth <= 0:
  #here the turtle draws the triangles
  for i in range(3):
   fd(t, l)
   lt(t, 120)
 else:
  for i in range(3):
   #here the turtle postions itself for the next triangle
   triangle(t, l, depth-1)
   fd(t, l*2**depth, depth-1) #depth here is the thickness of drawline
   lt(t, 120)
#positons the turtle at the begining
pu(turtle)
bk(turtle, 300)
lt(turtle)
bk(turtle, 250)
rt(turtle)
pd(turtle)

triangle(turtle, 70, 5)
wait_for_user()
```

6.2 Fractal tree

```python
from FractalWorld import *
world = FractalWorld(width=1900,height=1000)
turtle = Fractal(draw=True)
print turtle
set_pen_color(turtle,"Green")
color=["Brown","Blue","Green"]

def tree(t,l,a,depth):
 if depth==0:
#  print '1',depth
  return
 #experementing with colors
 if depth<4:
  set_pen_color(t,"#1a9108")
 elif depth==4:
  set_pen_color(t,"#d14500")
 elif depth==5:
  set_pen_color(t,"#b23b00")
 elif depth==6:
  set_pen_color(t,"#993401")
 else:
  set_pen_color(t,"#7A2900")
 pd(t)
 fd(t,l/0.666**depth,2*depth-1)
 rt(t,a)
# print '2',depth
 tree(t,l,a,depth-1)
# print '3',depth
 lt(t, 2*a)
 tree(t,l,a,depth-1)
# print '4',depth
 rt(t,a)
 pu(t)
 bk(t,l/0.666**depth)
# print '5',depth

pu(turtle)
lt(turtle)
bk(turtle, 450)
pd(turtle)

tree(turtle,5,30,18/3)
wait_for_user()
```

6.3 Class program

```
from FractalWorld import *
import sys

world = FractalWorld(width=800,height=800,delay=0)
t = Fractal(draw=True)

if sys.argv[1]=='tree.fdl':
  pu(t)
  lt(t,90)
  bk(t,200)
  pd(t)

class rule(object):
 def __init__(self):
  self.rule={}

 def rule1(self,left,right):
  self.rule[left]=right

 def udv(self,lista,depth):
  if depth<=0:
```

```python
   return lista
  liste=[]
  for i in lista:
   if i not in self.rule:
    liste.append(i)
   else:
    liste.append(self.udv(self.rule.get(i),depth-1))
  return liste

class command(object):
 def __init__(self,cmd,a):
  self.cmd=cmd
  self.a=a

 def udf(self,turtle,length):
  if self.cmd=="fd":
   fd(turtle,length[0])
  if self.cmd=="bk":
   bk(turtle,length[0])
  if self.cmd=="lt":
   lt(turtle,int(self.a[0]))
  if self.cmd=="rt":
   rt(turtle,int(self.a[0]))
  if self.cmd=="scale":
   length[0]=length[0]*float(self.a[0])

class fractal(object):
 def __init__(self,start,regler,commando,length,depth):
  self.start=start
  self.regler=regler
  self.commando=commando
  self.length=[length]
  self.depth=depth

 def tegn(self,lista):
  for i in lista:
   if type(i)==list:
    self.tegn(i)
   else:
    cmd=self.commando.get(i)
    cmd.udf(t,self.length)

def read():
  commando={}
  r=rule()
  filex = open(sys.argv[1])
  for line in filex:
   line=line.strip()
   lista=line.split(' ')
   if 'start' in line:
    start=lista[1:]
   elif 'length' in line:
    length=int(lista[1])
   elif 'rule' in line:
    r.rule1(lista[1],lista[3:])
   elif 'depth' in line:
    depth=int(lista[1])
   elif 'cmd' in line:
    cmd=command(lista[2],lista[3:])
    commando[lista[1]]=cmd
  return fractal(start,r,commando,length,depth)

frac=read()

a=frac.regler.udv(frac.start,frac.depth)
frac.tegn(a)

wait_for_user()
```