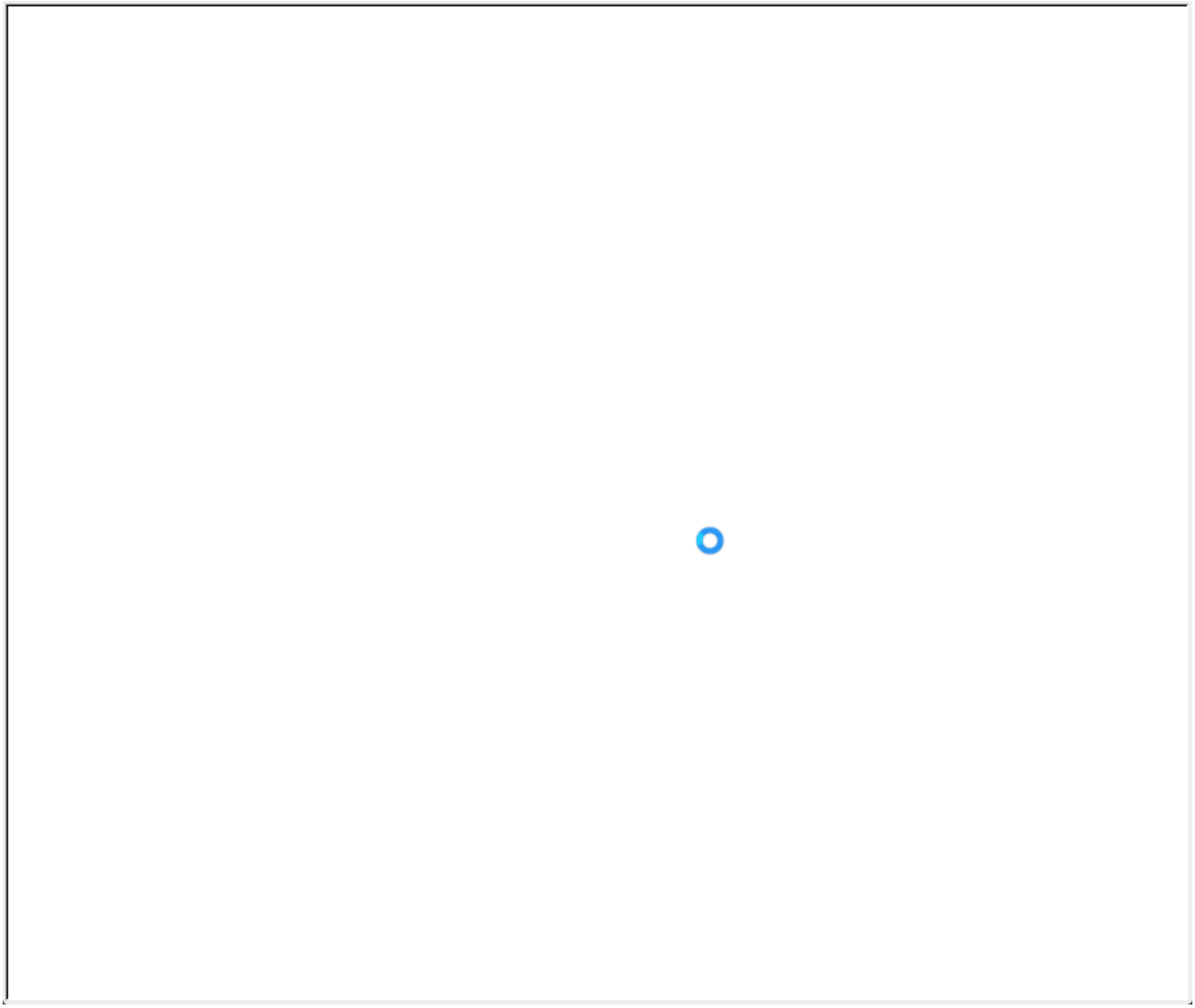
PY: Fraktale w pythonie

Wikipedia: L-systems

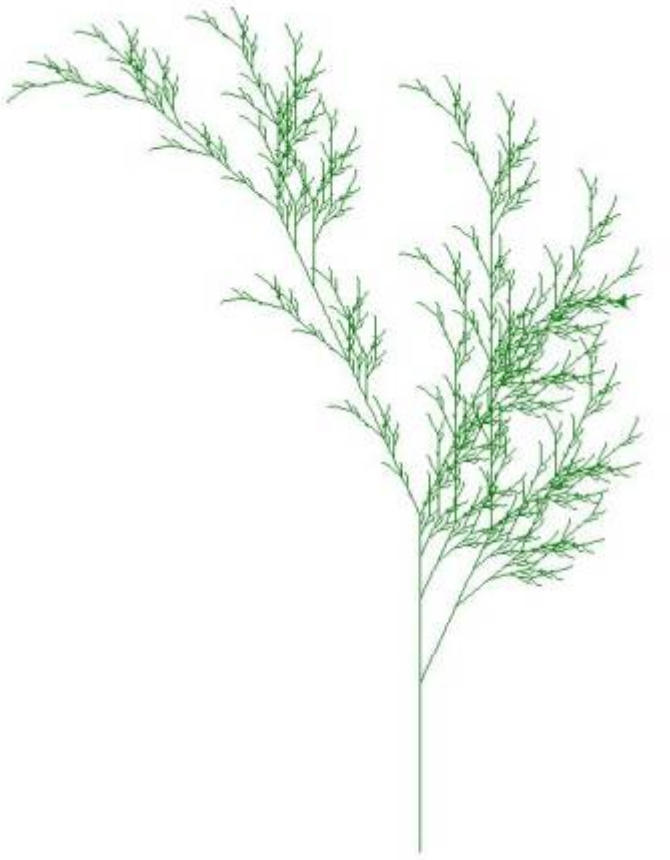


fractal_ardugeek.py

```
import turtle
import time

# Configuration
FONT = ("Arial", 48, "bold")
LETTER_SPACING = 65
ANIMATION_DELAY = 0.5  # seconds between drawing each letter
FRACTAL_ITER = 2       # reduced iterations for faster drawing
FRACTAL_SCALE = 10     # scaling factor for fractal drawing

# L-system definitions for each letter (axiom, rules, angle)
# 8 distinct fractals for 8 letters in "ArduGeek"
L_SYSTEMS = [
```

```
    ("F", {"F": "F+F--F+F"}, 60),                          # Koch curve
    ("FX", {"X": "X+YF+", "Y": "-FX-Y"}, 90),         # Dragon curve
    ("F-G-G", {"F": "F-G+F+G-F", "G": "GG"}, 120),     # Sierpinski
triangle
    ("X", {"X": "F-[[X]+X]+F[+FX]-X", "F": "FF"}, 25),# Fractal plant
    ("A", {"A": "B-A-B", "B": "A+B+A"}, 60),          # Arrowhead curve
    ("X", {"X": "X+YF++YF-FX--FXFX-YF+", "Y": "-FX+YFYF++YF+FX--FX-Y"},
90), # Gosper curve
    ("F+F+F+F", {"F": "F+F-F-F+F"}, 90),                 # Smaller
square Koch variant
    ("F", {"F": "F+F-F"}, 120) # Terdragon curve for an intricate,
compact design
]

# Setup screen and turtles
def setup():
    screen = turtle.Screen()
    screen.title("ArduGeek with Persistent Fractals")
    screen.bgcolor("white")

    pen = turtle.Turtle()
    pen.hideturtle()
    pen.penup()
    pen.speed(1)
    pen.color("darkblue")

    fractal_t = turtle.Turtle()
    fractal_t.hideturtle()
    fractal_t.penup()
    fractal_t.speed(0)
    fractal_t.color("gray")

    return screen, pen, fractal_t

# Generate L-system string
def generate_lsystem(axiom, rules, iterations):
    s = axiom
    for _ in range(iterations):
        s = ''.join(rules.get(ch, ch) for ch in s)
    return s

# Draw L-system with given turtle
def draw_lsystem(t, instructions, angle, scale):
    for cmd in instructions:
        if cmd in ('F', 'G'):
            t.forward(scale)
        elif cmd == '+':
            t.right(angle)
        elif cmd == '-':
            t.left(angle)
        elif cmd == '[':
```

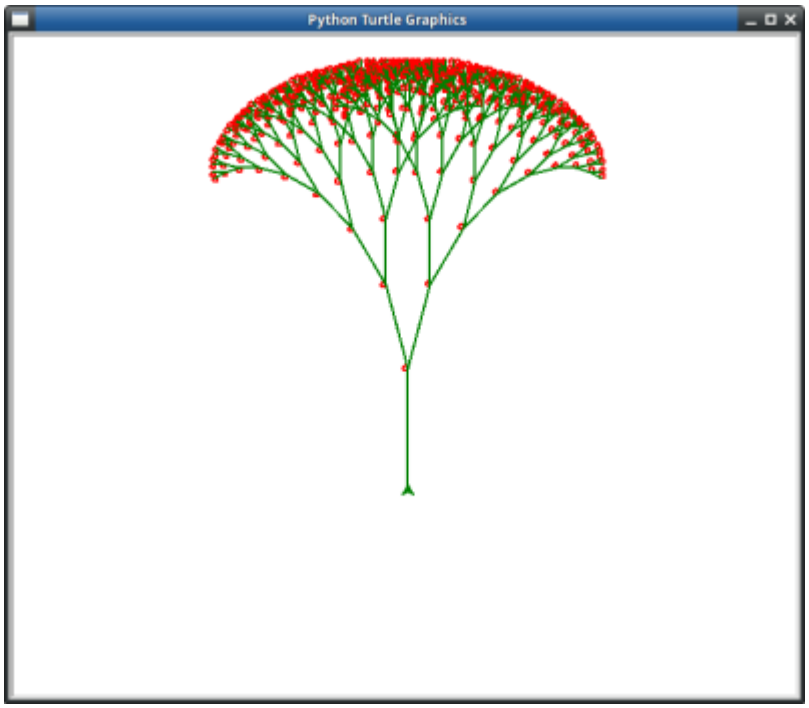
```
            stack.append((t.position(), t.heading()))
        elif cmd == ']':
            pos, head = stack.pop()
            t.penup()
            t.goto(pos)
            t.setheading(head)
            t.pendown()

# Main animation: draw each fractal once, then write letters on top
def animate_text_with_fractals(screen, pen, fractal_t, text):
    # center the text on screen
    total_width = len(text) * FONT[1] * 0.6 + (len(text)-1) *
LETTER_SPACING
    start_x = -total_width / 2
    baseline_y = 0
    pen.goto(start_x, baseline_y)

    # draw persistent fractals behind each letter
    for i, letter in enumerate(text):
        axiom, rules, angle = L_SYSTEMS[i]
        inst = generate_lsystem(axiom, rules, FRACTAL_ITER)
        fractal_t.penup()
        # align each fractal under its letter
        fractal_t.goto(start_x + i*(FONT[1]*0.6 + LETTER_SPACING),
baseline_y - FONT[1]*0.6)
        fractal_t.setheading(90)  # align fern and others upright
        fractal_t.pendown()
        global stack
        stack = []
        draw_lsystem(fractal_t, inst, angle, FRACTAL_SCALE)
        fractal_t.penup()

    # write all letters on top, one by one
    pen.goto(start_x, baseline_y)
    for letter in text:
        pen.write(letter, font=FONT, align="left")
        pen.forward(FONT[1]*0.6 + LETTER_SPACING)
        time.sleep(ANIMATION_DELAY)

if __name__ == '__main__':

    screen, pen, fractal_t = setup()
    input()
    animate_text_with_fractals(screen, pen, fractal_t, "ArduGeek")
    pen.penup()
    screen.mainloop()
```



fractal_plant.py

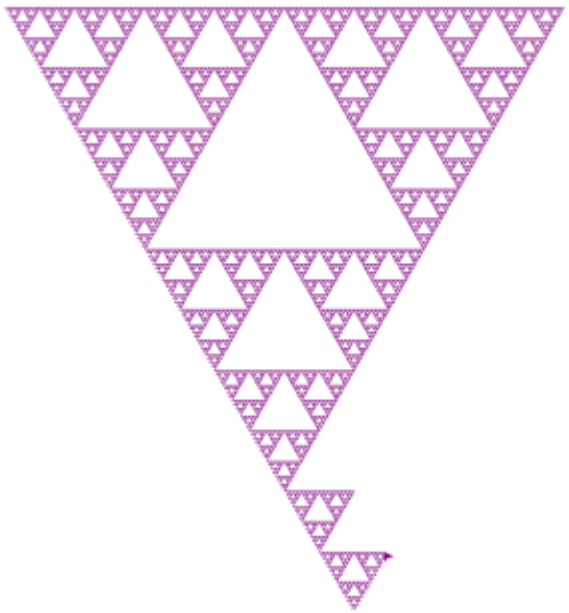
```
from turtle import *

tracer(0)
start="X"
dlugosc=4
kat=25

stos=[]
slZam={'X':'F+[[X]-X]-F[-FX]+X','F':'FF'}

iteracje=6
zolw='zolw'

def LSBuduj(st,ile,sl):
    nowy=""
    for litera in st:
        if litera in st:
            if litera in sl.keys():
                nowy+=sl[litera]
            else:
                nowy+=litera

    if ile>1:
        ile-=1
```

```
        return LSBuduj(nowy,ile,sl)
    else:
        return nowy
DoWykonania=LSBuduj(start,iteracje,slZam)
Polecenia={}
Polecenia['F']=[zolw+'.pd()',zolw+'.fd('+str(dlugosc)+')']
Polecenia['+']=[zolw+'.right('+str(kat)+')']
Polecenia['-']=[zolw+'.left('+str(kat)+')']
Polecenia['[']=['stos.append(('+zolw+'.xcor(),'+zolw+'.ycor(),'+zolw+'.
heading()))']
Polecenia[']']=[zolw+'.pu()',zolw+'.setx(stos[len(stos)-1][0])',
                            zolw+'.sety(stos[len(stos)-1][1])',
                            zolw+'.setheading(stos[len(stos)-1][2])',
                            'stos.pop()']

print(Polecenia)

zolw=Turtle()
zolw.pu()
zolw.goto(0,-300)
zolw.color('green')
zolw.pd()
zolw.setheading(90)
zolw.speed(0)
for litera in DoWykonania:
    if litera in Polecenia.keys():
        for rozkaz in Polecenia[litera]:
            eval(rozkaz)
update()
```



binary_tree.py

```
import turtle as t
t.speed(0)
t.pensize(2)
t.left(90)
t.backward(100)
t.color("green")

def draw(l):
    if(l<10):
        return
    else:
        t.forward(l)
        t.color("red")
        t.circle(2)
        t.color("green")
        t.left(45)
        draw(3*l/4)
        t.right(90)
        draw(3*l/4)
        t.left(45)
        t.backward(l)

draw(25)
t.exitonclick()
```



siepinsky_triangle.py

```
from turtle import *


start="F-G-G"
```

```
dlugosc=5
kat=120

slownik={}
slownik['G']="GG"
slownik['F']="F-G+F+G-F"

iteracje=10
zolw='zolw'

def LSBuduj(st,ile,sl):
    nowy=""
    for litera in st:
        if litera in slownik.keys():
            nowy+=sl[litera]
        else:
            nowy+=litera

    if ile>1:
        ile-=1
        return LSBuduj(nowy,ile,sl)
    else:
        return nowy

#print(len(LSBuduj(start,iteracje,slownik)))

DoWykonania=LSBuduj(start,iteracje,slownik)


Polecenia={}
Polecenia["G"]=[zolw+".fd("+str(dlugosc)+")"]
Polecenia["F"]=[zolw+".fd("+str(dlugosc)+")"]
Polecenia["+"]=[zolw+".left("+str(kat)+")"]
Polecenia["-"]=[zolw+".right("+str(kat)+")"]



zolw=Turtle()
zolw.pu()
zolw.goto(-300,200)
zolw.color('purple')
zolw.pd()
zolw.speed(0)
for litera in DoWykonania:
    if litera in Polecenia.keys():
        for rozkaz in Polecenia[litera]:
            eval(rozkaz)
```