# Python: programming

Asst. Prof. Dr. Yalçın İŞLER

# Loops: break

```
#!/usr/bin/python

for letter in 'Python':     # First Example
   if letter == 'h':
      break
   print 'Current Letter :', letter

var = 10                    # Second Example
while var > 0:
   print 'Current variable value :', var
   var = var -1
   if var == 5:
      break

print "Good bye!"
```

```
Current Letter : P
Current Letter : y
Current Letter : t
Current variable value : 10
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Good bye!
```

# Modules: define and use

support.py

```
def print_func( par ):
   print "Hello : ", par
   return
```

```
#!/usr/bin/python

# Import module support
import support

# Now you can call defined function that module as follows
support.print_func("Zara")
```

```
Hello : Zara
```

# Modules: import options

```
from modname import name1[, name2[, ... nameN]]
```

```
from fib import fibonacci
```

to import the function of fibonacci from the module of fib

```
from modname import *
```

to import everything including definitions and functions from the module of fib

# Scope: Use Global vs. Local

```
#!/usr/bin/python

Money = 2000
def AddMoney():
    # Uncomment the following line to fix the code:
    # global Money
    Money = Money + 1

print Money
AddMoney()
print Money
```

# Conditionals: if else vs. nested if

```
num = float(input("Enter a number: "))
if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

if – elif – else structure

```
Enter a number: 2
Positive number
```

```
num = float(input("Enter a number: "))
if num >= 0:
    if num == 0:
        print("Zero")
    else:
        print("Positive number")
else:
    print("Negative number")
```

nested if structure

# Example: for

```
# Python program to check if the input number is prime or not

num = 407

# take input from the user
# num = int(input("Enter a number: "))

# prime numbers are greater than 1
if num > 1:
   # check for factors
   for i in range(2,num):
       if (num % i) == 0:
           print(num,"is not a prime number")
           print(i,"times",num//i,"is",num)
           break
   else:
       print(num,"is a prime number")

# if input number is less than
# or equal to 1, it is not prime
else:
   print(num,"is not a prime number")
```

```
407 is not a prime number
11 times 37 is 407
```

# Example: normal vs recursive

```python
# Python program to find the factorial of a number provided by the user.

# change the value for a different result
num = 7

# uncomment to take input from the user
#num = int(input("Enter a number: "))

factorial = 1

# check if the number is negative, positive or zero
if num < 0:
   print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
   print("The factorial of 0 is 1")
else:
   for i in range(1,num + 1):
       factorial = factorial*i
   print("The factorial of",num,"is",factorial)
```

```
The factorial of 7 is 5040
```

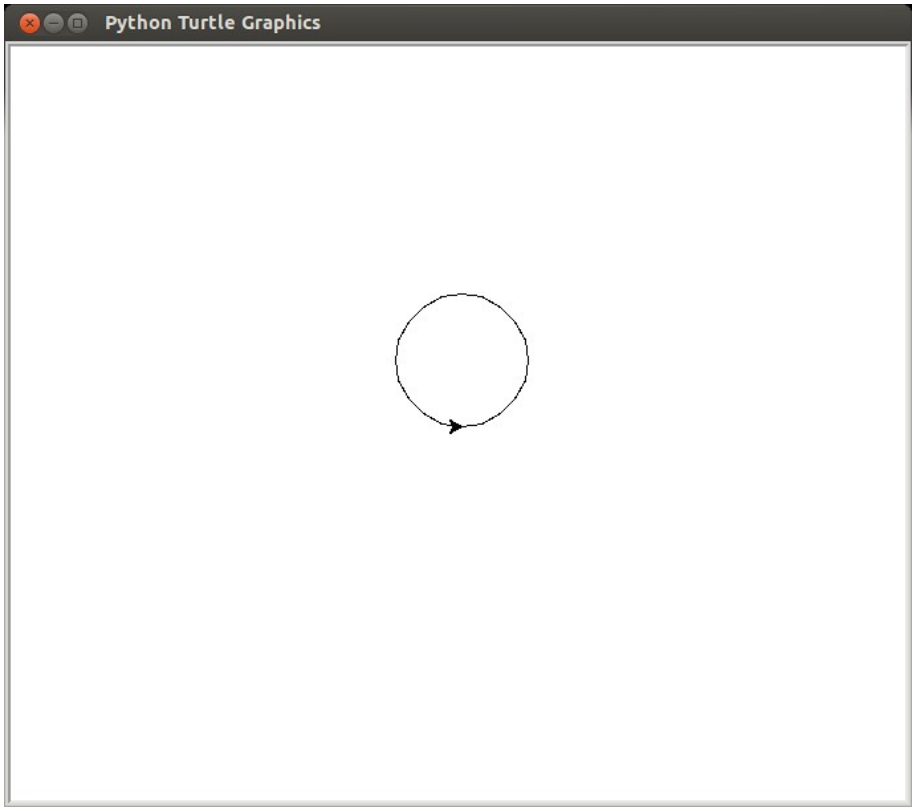
# Example: normal vs recursive

```
# Python program to find the factorial of a number using recursion

def recur_factorial(n):
    """Function to return the factorial
    of a number using recursion"""
    if n == 1:
        return n
    else:
        return n*recur_factorial(n-1)

# Change this value for a different result
num = 7

# uncomment to take input from the user
#num = int(input("Enter a number: "))

# check is the number is negative
if num < 0:
    print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
    print("The factorial of 0 is 1")
else:
    print("The factorial of",num,"is",recur_factorial(num))
```

```
The factorial of 7 is 5040
```

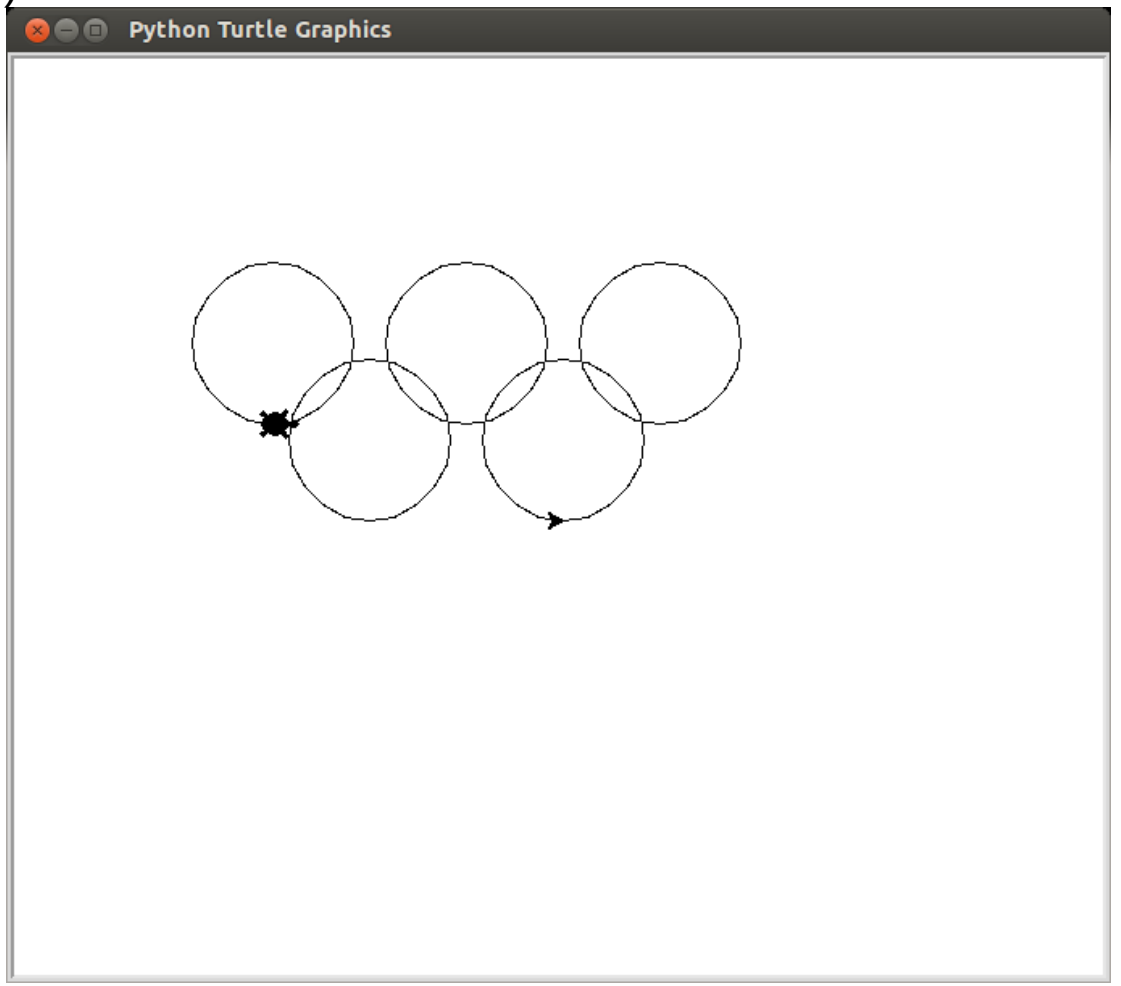
# Graphics 1: turtle

```
import turtle

# ornekliyoruz
kaplumbagam = turtle.Turtle()

# 50 derece cember ciziyoruz
kaplumbagam.circle(50)

# cemberi ekranda tutuyoruz
turtle.getscreen()._root.mainloop()
```



# Graphics 2: turtle

```
import turtle

kaplumbagam = turtle.Turtle(shape="turtle")
kaplumbagam.circle(50)

kaplumbagam.penup()
kaplumbagam.setposition(-120, 0)
kaplumbagam.pendown()
kaplumbagam.circle(50)

kaplumbagam.penup()
kaplumbagam.setposition(60,60)
kaplumbagam.pendown()
kaplumbagam.circle(50)

kaplumbagam.penup()
kaplumbagam.setposition(-60, 60)
kaplumbagam.pendown()
kaplumbagam.circle(50)

kaplumbagam.penup()
kaplumbagam.setposition(-180, 60)
kaplumbagam.pendown()
kaplumbagam.circle(50)

turtle.getscreen()._root.mainloop()
```



# Graphics 3: turtle

```
#!/usr/bin/env python3

import sys
import turtle
```

```
def square(t, size, color):
    """(Turtle, int, str)

    Draw a square of the chosen colour and size.
    """
    t.pencolor(color)
    t.pendown()
    for i in range(4):
        t.forward(size)
        t.right(90)
```

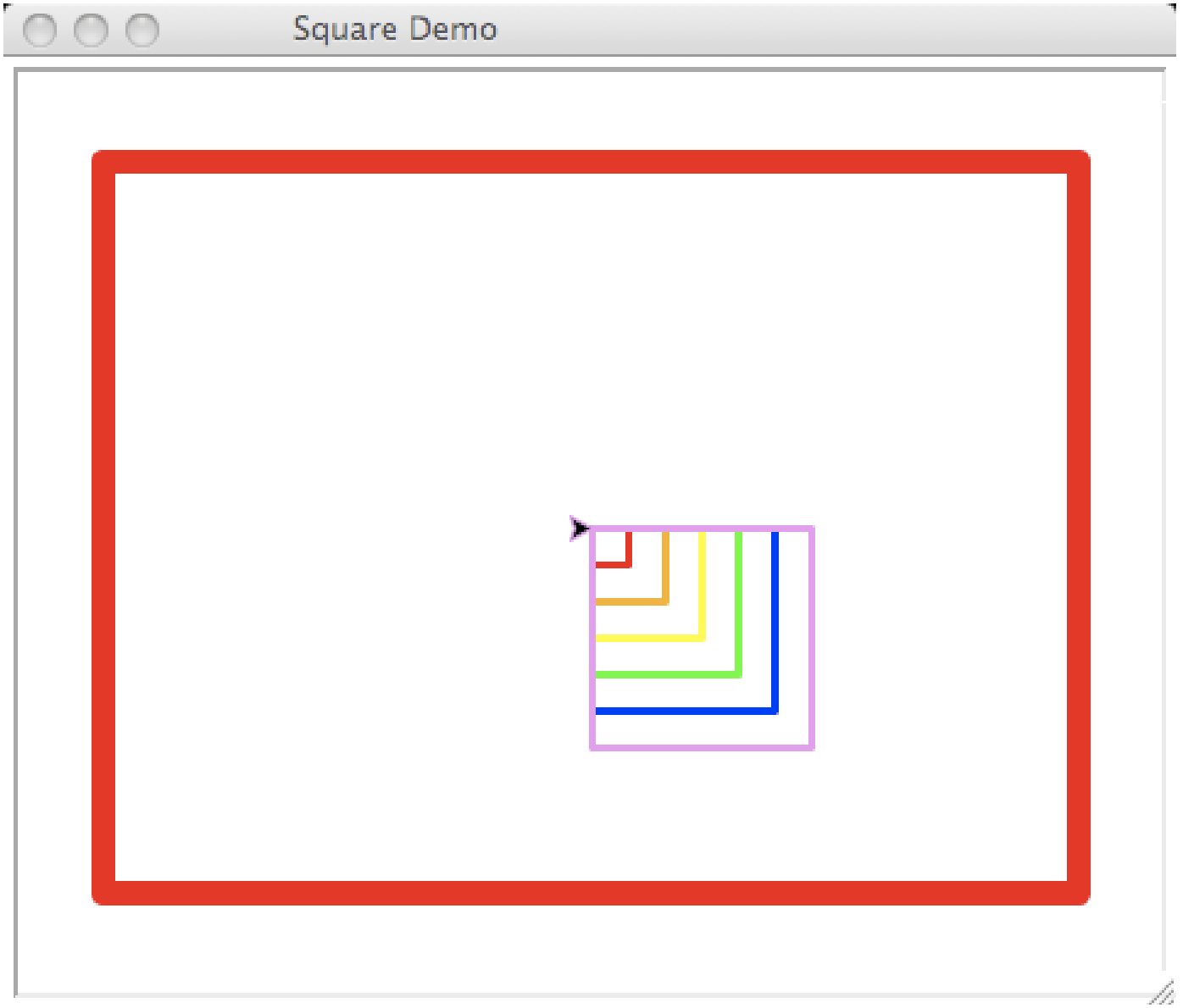
# Graphics 3: turtle (cont'd)

```
def border(t, screen_x, screen_y):
    """(Turtle, int, int)

    Draws a border around the canvas in red.
    """
    # Lift the pen and move the turtle to the center.
    t.penup()
    t.home()

    # Move to lower left corner of the screen; leaves the turtle
    # facing west.
    t.forward(screen_x / 2)
    t.right(90)
    t.forward(screen_y / 2)
    t.setheading(180)           # t.right(90) would also work.

    # Draw the border
    t.pencolor('red')
    t.pendown()
    t.pensize(10)
    for distance in (screen_x, screen_y, screen_x, screen_y):
        t.forward(distance)
        t.right(90)

    # Raise the pen and move the turtle home again; it's a good idea
    # to leave the turtle in a known state.
    t.penup()
    t.home()
```

# Graphics 3: turtle (cont'd)

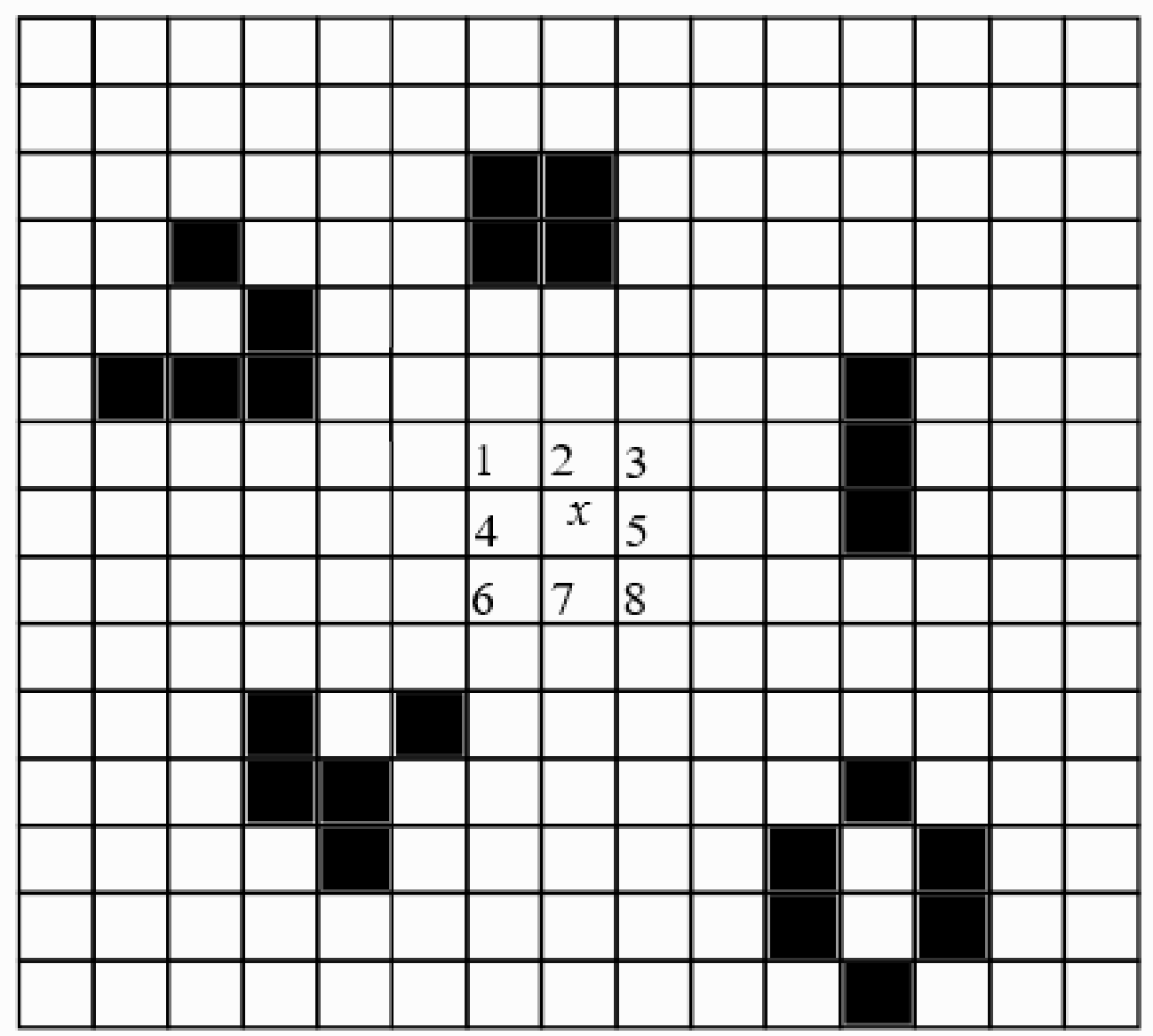
```
def main():
    # Create screen and turtle.
    screen = turtle.Screen()
    screen.title('Square Demo')
    screen_x, screen_y = screen.screensize()
    t = turtle.Turtle()

    # Uncomment to draw the graphics as quickly as possible.
    ##t.speed(0)

    # Draw a border around the canvas
    border(t, screen_x, screen_y)

    # Draw a set of nested squares, varying the color.
    # The squares are 10%, 20%, etc. of half the size of the canvas.
    colors = ['red', 'orange', 'yellow', 'green', 'blue', 'violet']
    t.pensize(3)
    for i, color in enumerate(colors):
        square(t, (screen_y / 2) / 10 * (i+1), color)

    print('Hit any key to exit')
    dummy = input()

if __name__ == '__main__':
    main()
```

# Graphics 3: turtle (cont'd)



# Graphics 4: turtle

# Game: Live Cell



# Game: Live Cell Game rules

If the cell is dead:

1. **Birth**: if exactly three of its neighbours are alive, the cell will become alive at the next step.

If the cell is already alive:

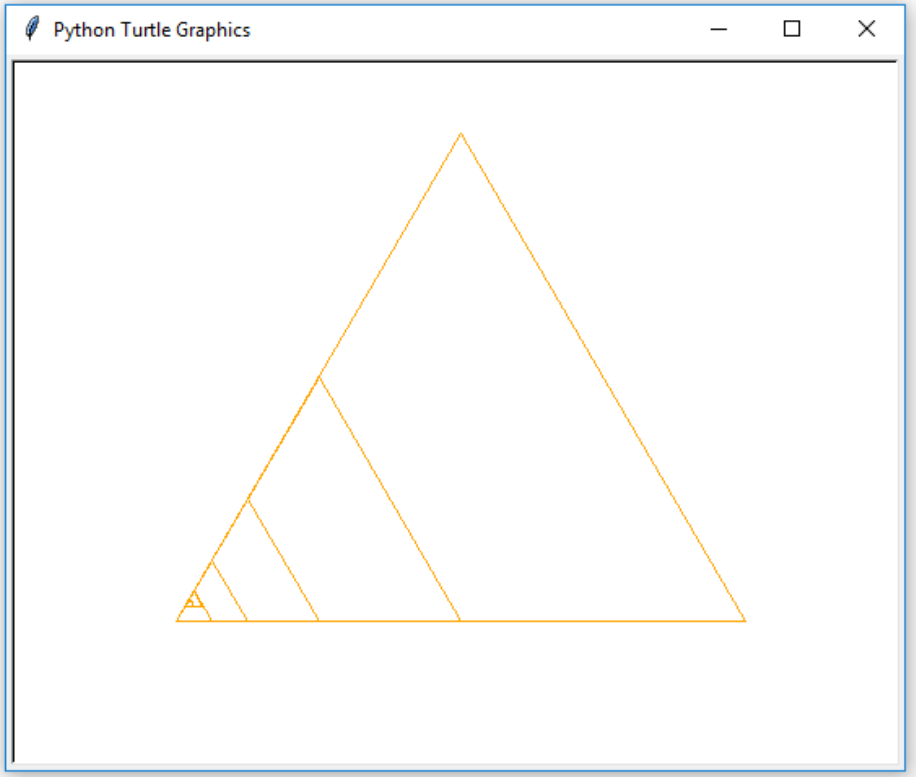
1. **Survival**: if the cell has two or three live neighbours, the cell remains alive.

Otherwise, the cell will die:

2. **Death by loneliness**: if the cell has only zero or one live neighbours, the cell will become dead at the next step.
3. **Death by overcrowding**: if the cell alive and has more than three live neighbours, the cell also dies.

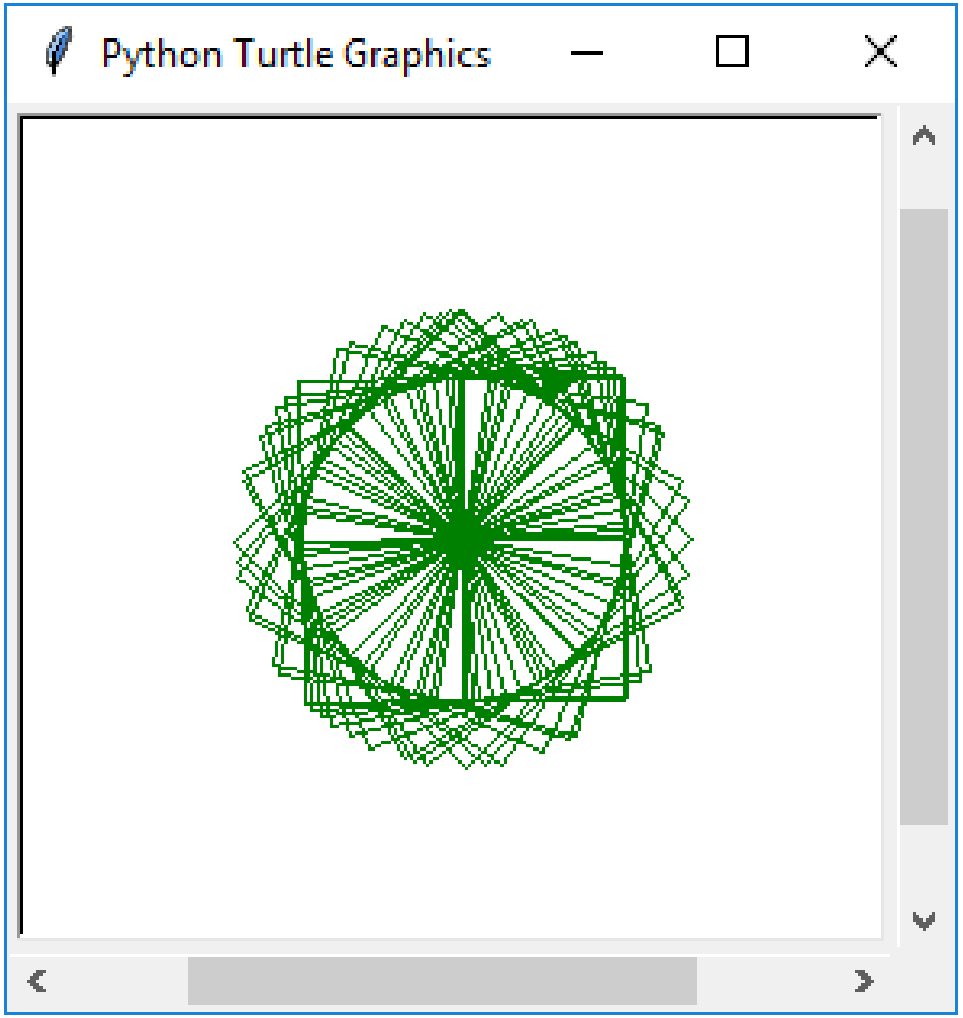
# Fractal 1: Triangles

```
import turtle
PROGNAME = 'Sierpinski Triangle'

myPen = turtle.Turtle()
myPen.ht()
myPen.speed(5)
myPen.pencolor('orange')

points = [[-175,-125],[0,175],[175,-125]] #size of triangle

def getMid(p1,p2):
  return ( (p1[0]+p2[0]) / 2, (p1[1] + p2[1]) / 2) #find midpoint

def triangle(points,depth):
  myPen.up()
  myPen.goto(points[0][0],points[0][1])
  myPen.down()
  myPen.goto(points[1][0],points[1][1])
  myPen.goto(points[2][0],points[2][1])
  myPen.goto(points[0][0],points[0][1])

  if depth>0:
    triangle([points[0],
    getMid(points[0], points[1]),
    getMid(points[0], points[2])],
    depth-1)
  triangle([points[1],
    getMid(points[0], points[1]),
    getMid(points[1], points[2])],
    depth-1)
  triangle([points[2],
    getMid(points[2], points[1]),
    getMid(points[0], points[2])],
    depth-1)

triangle(points,4)
```



# Fractal 2: Circle from Squares

```
import turtle
import time

def draw_square():
    turtle.forward(50)
    turtle.left(90)
    turtle.forward(50)
    turtle.left(90)
    turtle.forward(50)
    turtle.left(90)
    turtle.forward(50)
    turtle.left(90)

turtle.color('green')
for i in range(360):
    draw_square()
    turtle.left(i)
```



# Fractal 3: Tree

```
import turtle

myTurtle = turtle.Turtle()

def tree(t,length,n):

  if length < (length/n):
    return
  t.forward(length)
  t.left(45)
  tree(t, length * 0.5,length/n)
  t.left(20)
  tree(t, length * 0.5,length/n)
  t.right(75)
  tree(t, length * 0.5,length/n)
  t.right(20)
  tree(t, length * 0.5,length/n)
  t.left(30)
  t.backward(length)
  return

myTurtle.left(90)
myTurtle.backward(30)
tree(myTurtle,200,4)
```