Temă: metoda transformărilor iterate

1) Incercaţi să desenaţi C*urba lui Peano*[1], a cărei construcţie poate fi dedusă din primele două etape indicate în figurile 1 şi 2.

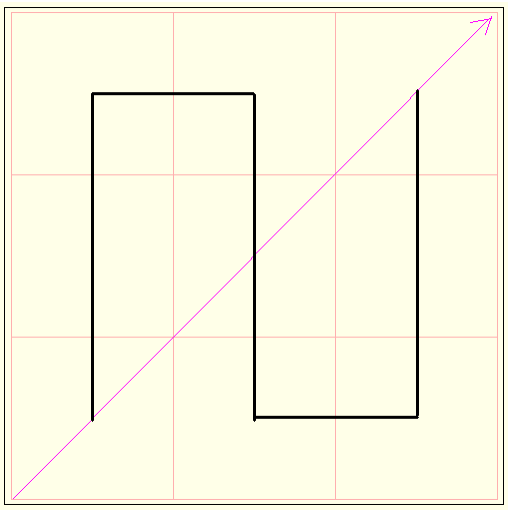


Figura 1.

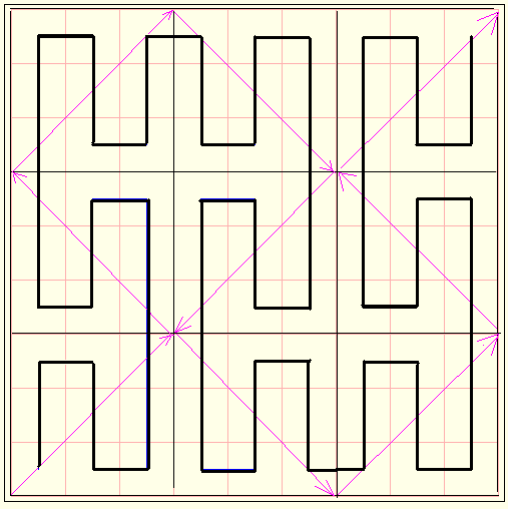


Figura 2.

1 vezi https://en.wikipedia.org/wiki/Peano_curve

Indicaţie: Programul următor trasează desenul din Figura 3:

```
import ComplexPygame as C
import Color
def Peano():
    c1 = (1 + 1j) / 6
    c2 = (1 + 3j) / 6
    c3 = (1 + 5j) / 6
    c4 = (3 + 5j) / 6
    c0 = (3 + 3j) / 6
    c5 = (3 + 1j) / 6
    c6 = (5 + 1j) / 6
    c7 = (5 + 3j) / 6
    c8 = (5 + 5j) / 6

    def s0(z):
        return c0 + (z - c0) / 3

    def s1(z):
        return c1 + (z - c0) / 3

    def s2(z):
        return c2 + (z - c0) / 3

    def s3(z):
        return c3 + (z - c0) / 3

    def s4(z):
        return c4 + (z - c0) / 3

    def s5(z):
        return c5 + (z - c0) / 3

    def s6(z):
        return c6 + (z - c0) / 3

    def s7(z):
        return c7 + (z - c0) / 3

    def s8(z):
        return c8 + (z - c0) / 3

    def transforma(li):
        rez = []
        for z in li:  rez.append(s1(z))
        for z in li:  rez.append(s2(z))
        for z in li:  rez.append(s3(z))
        for z in li:  rez.append(s4(z))
        for z in li:  rez.append(s0(z))
        for z in li:  rez.append(s5(z))
        for z in li:  rez.append(s6(z))
        for z in li:  rez.append(s7(z))
```

```
        for z in li:  rez.append(s8(z))
        return rez

    def traseaza(li):
        C.fillScreen()
        # trasam chenarul
        C.drawNgon([0, 1, 1 + 1j, 1j], Color.Black)
        for n in range(1, len(li)):
            col = Color.Red if n % 9 == 0 else Color.Blue
            C.drawLine(li[n - 1], li[n], col)

    C.setXminXmaxYminYmax(-0.1, 1.1, -0.1, 1.1)
    fig = [c0]
    for k in range(2):
        fig = transforma(fig)
        traseaza(fig)
        if C.mustClose(): return

if __name__ == '__main__':
    C.initPygame()
    C.run(Peano)
```

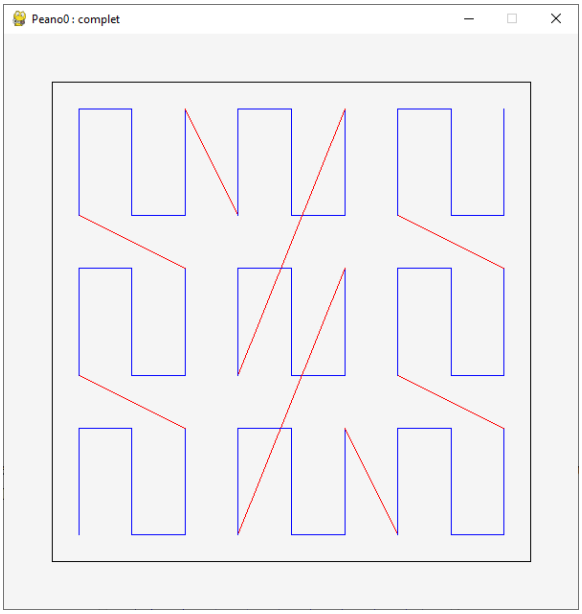


Figura 3

Mai trebuie doar corectate unele dintre transformările **`s0, s1, … ,s8`** astfel încât rezultatul să arate ca în Figura 4:

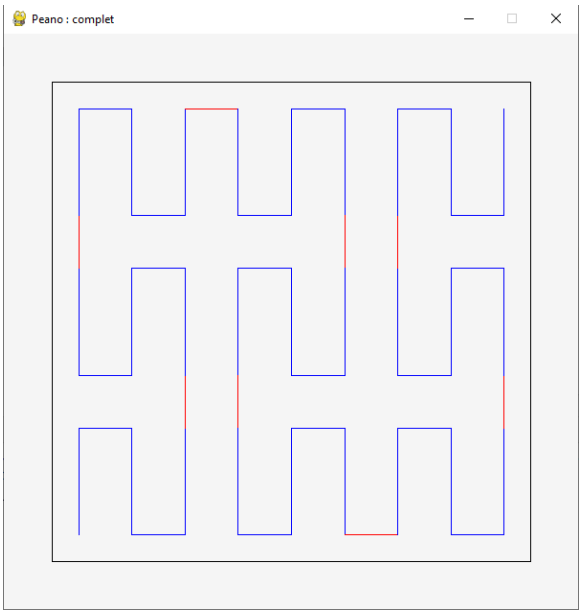
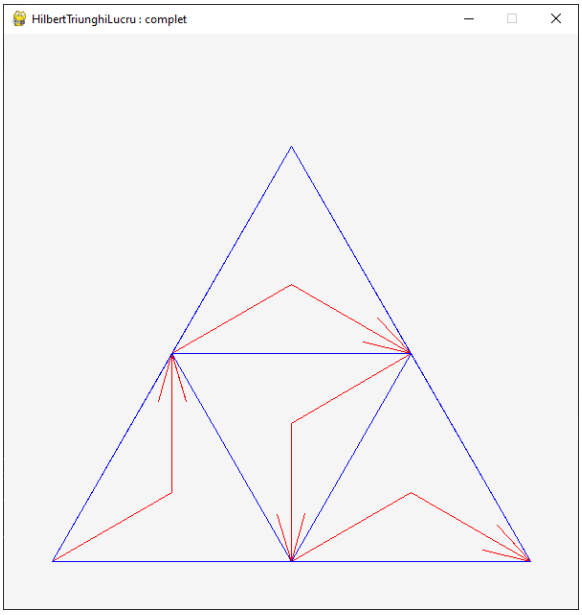


Figura 4.

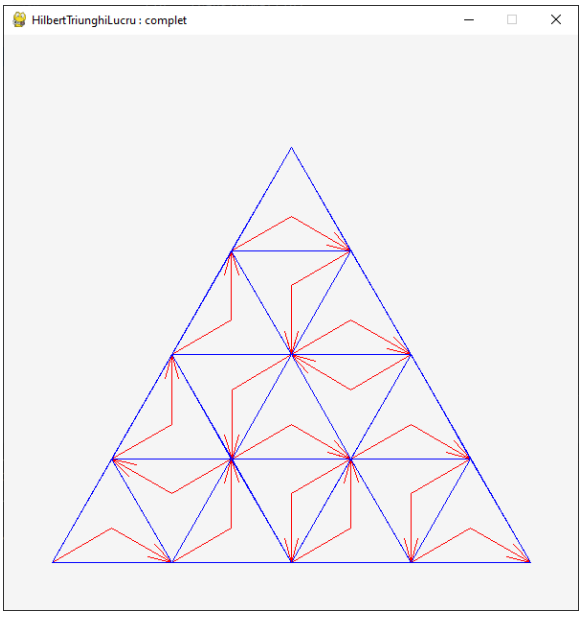
2) Incercaţi să umpleţi un triunghi parcurgându-l în modul sugerat de următoarele primele trei etape:

Etapa 1:

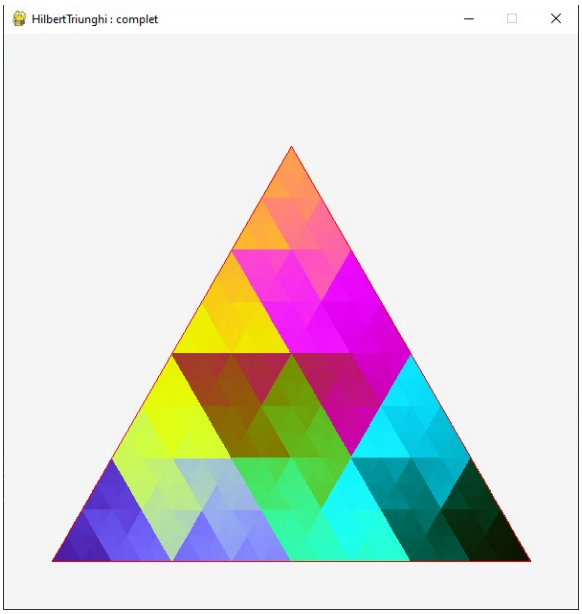
Etapa 2:



Etapa 3:



Rezultat, după ce s-a schimbat modul de colorare:

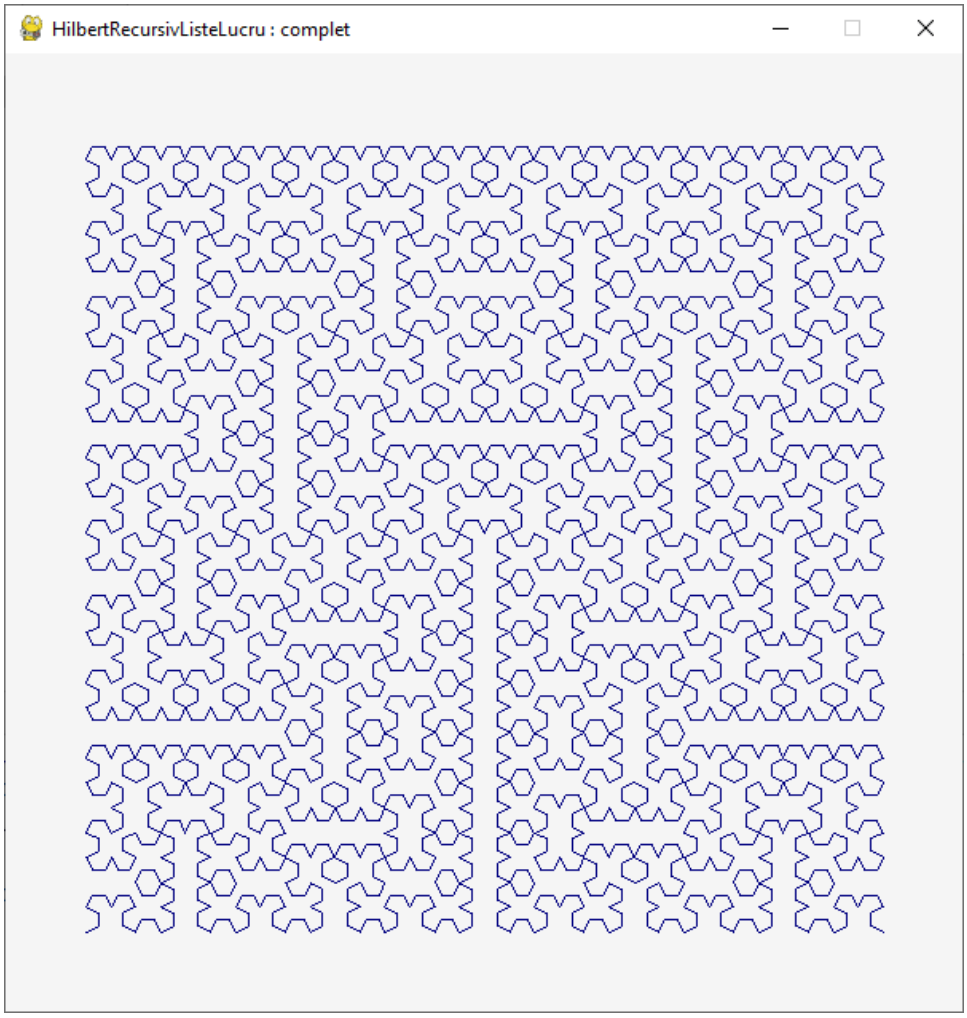


3) Următorul program trasează *Curba lui Hilbert* printr-o implementare cu funcţii recursive:

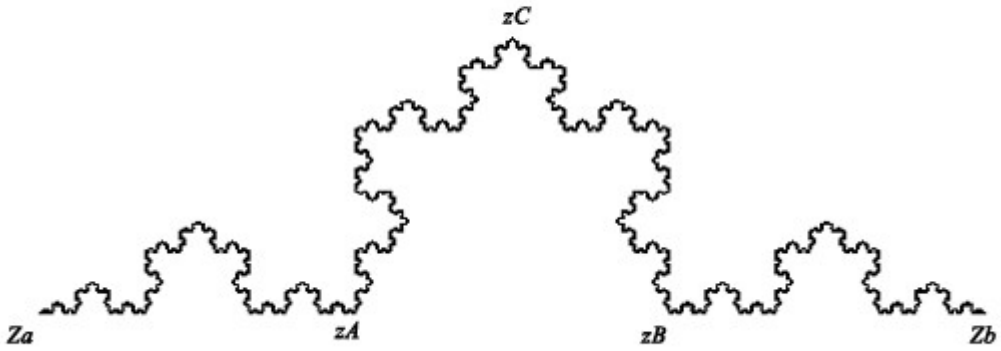
```
import ComplexPygame as C
import Color
def HilbertRecursivListe():
    lista = []

    #    // traseu prin patrat:
    #    // z0 = coltul din stanga jos
    #    // z0+d1 = stanga sus
    #    // z0+d1+d2 = dreapta sus
    #    // z0+d2 = dreapta jos
    def genereaza(z0, d1, d2, niv):
        d1 *= 0.5
        d2 *= 0.5
        niv -= 1
        if niv < 0:
            lista.append(z0 + d1 + d2)
        else:
            genereaza(z0, d2, d1, niv)
            genereaza(z0 + d1, d1, d2, niv)
            genereaza(z0 + d1 + d2, d1, d2, niv)
            genereaza(z0 + d1 + d2 + d2, -d2, -d1, niv)
        return

    C.setXminXmaxYminYmax(-0.1, 1.1, -0.1, 1.1)
    z0 = 0
    delta1 = 1j
    delta2 = 1
```

```
    nrEtape = 5
    genereaza(z0, delta1, delta2, nrEtape)
    C.fillScreen(Color.Navy)
    for k in range(1, len(lista)):
        C.drawLine(lista[k - 1], lista[k], Color.Yellow)
    C.refreshScreen()


if __name__ == '__main__':
    C.initPygame()
    C.run(HilbertRecursivListe)
```

Modificaţi codul executat de metoda `genereaza()` la ieşirea din recursivitate (pe ramura *true* a if-ului), astfel încât după 5 etape să apară următorul rezultat:



4) Trasaţi curba lui Koch prin metoda transformărilor geometrice iterate, folosind asemănarea dintre curba construită pe segmentul de bază $z_a z_b$ şi cele 4 curbe Koch construite pe segmentele $z_a z_A$, $z_A z_C$, $z_C z_B$ şi $z_B z_b$ ale motivului.

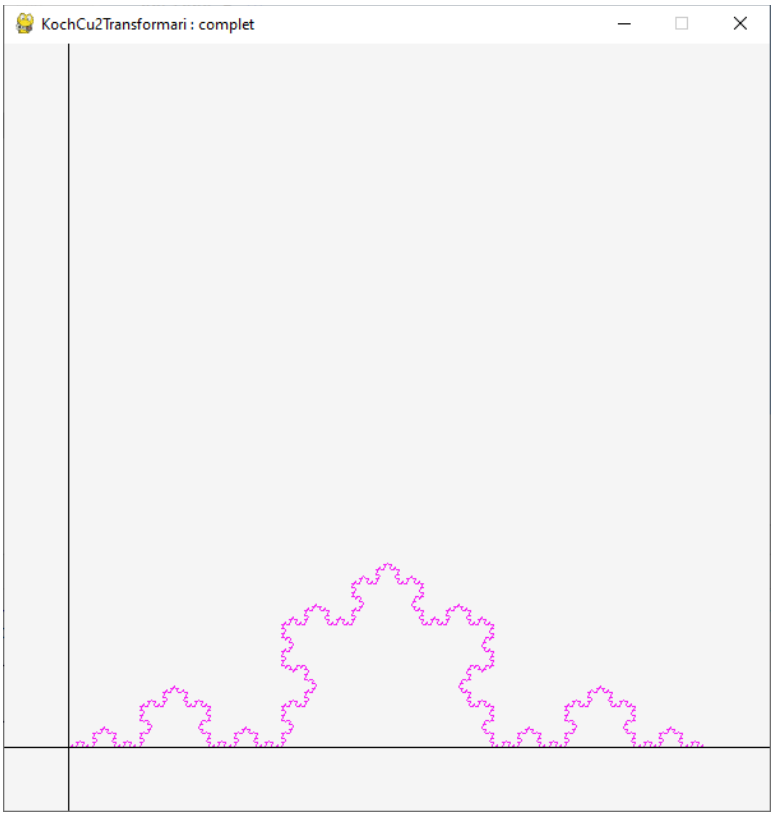


5) Următorul program trasează curba lui Koch cu două transformărilor geometrice iterate, folosind asemănarea dintre curba construită pe stânga segmentului de bază $z_a z_b$ şi cele două curbe Koch construite pe dreapta segmentelor $z_a z_C$ şi $z_C z_b$. Justificaţi forma transformărilor folosite.

Indicaţie: triunghiul Δabc este *invers-asemenea* cu Δpqr dacă triunghiul Δabc este *direct-asemenea* cu triunghiul format de conjugatele numerelor complexe p, q şi r, în această ordine.

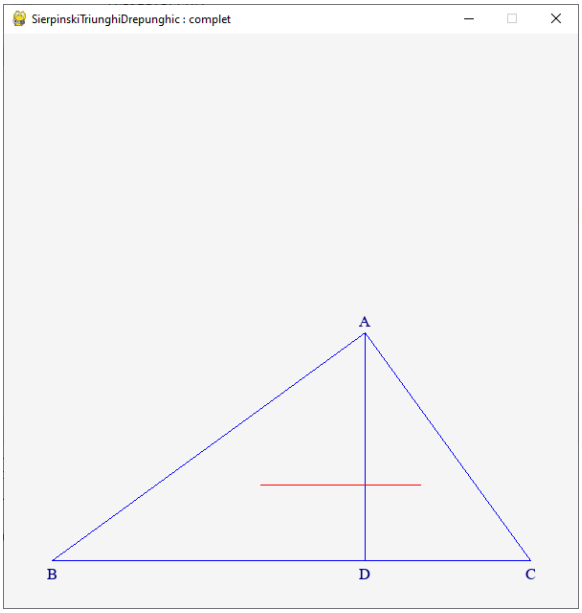
```
import ComplexPygame as C
import Color
import math


def KochCu2Transformari():
    theta = math.pi / 6
    rho = 0.5 / math.cos(theta)
    w = C.fromRhoTheta(rho, theta)
    zA = 0
    zB = 1
    zC = zA + w * (zB - zA)
    omega1 = (zC - zA) / (zB - zA).conjugate()
    omega2 = (zC - zB) / (zA - zB).conjugate()

    def T1(z):
        return zA + omega1 * (z - zA).conjugate()

    def T2(z):
        return zB + omega2 * (z - zB).conjugate()
```

```
    def transforma(li):
        rez = [T1(z) for z in li]
        rez.extend([T2(z) for z in li])
        return rez

    C.setXminXmaxYminYmax(-0.1, 1.1, -0.1, 1.1)
    fig = [zA, zB]
    nrEtape = 10
    for k in range(nrEtape):
        fig = transforma(fig)
        C.fillScreen()
        col = Color.Index(300 + 10 * k)
        for h in range(1, len(fig)):
            C.drawLine(fig[h - 1], fig[h], col)
        if C.mustClose(): return
        C.wait(50)
    C.setAxis()


if __name__ == '__main__':
    C.initPygame()
    C.run(KochCu2Transformari)
```



6) Orice triunghi dreptunghic $\Delta ABC(A=90°)$ este împărţit de înălţimea AD dusă din unghiul drept în două triunghiuri asemenea cu triunghiul iniţial.

Pornim cu un astfel de triunghi şi la fiecare etapă împărţim triunghiurile obţinute în două alte triunghiuri prin ducerea înălţimii. Linia poligonală formată de centrele de greutate ale triunghiurilor obţinute la etapa *n* reprezintă aproximarea de ordin *n* a *Curbei lui Sierpinski* într-un triunghi dreptunghic oarecare (cazul clasic este dat de triunghiul dreptunghic isoscel).



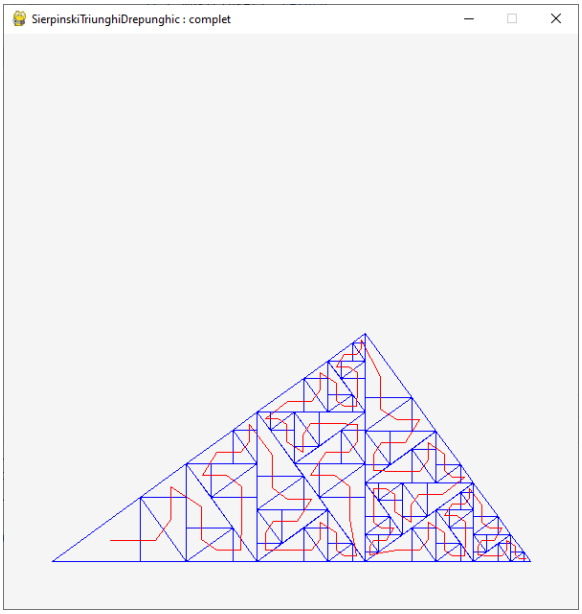
Următorul program trasează prin metoda transformărilor geometrice această curbă:

```
import ComplexPygame as C
import Color
import math
def SierpinskiTriunghiDrepunghic():
    zB = 0
    zC = 1
    zQ = (zB + zC) / 2
    zA = zQ + C.fromRhoTheta(C.rho(zC - zQ), 2 * math.pi / 5)
    k1 = (zA - zC) / (zB - zC).conjugate()
    k2 = (zA - zB) / (zC - zB).conjugate()

    def s1(z):
        return zC + k1 * (z - zC).conjugate()

    def s2(z):
        return zB + k2 * (z - zB).conjugate()

    def transforma(li):
        rez = [s1(z)for z in li]
        rez.extend([s2(z)for z in li])
        return rez
```

```
    def traseaza(li):
        C.fillScreen()
        zA = sum(li[0:3]) / 3
        C.drawNgon(li[0:3], Color.Blue)
        for n in range(6, len(li)+1, 3):
            zB = sum(li[n - 3:n]) / 3
            C.drawNgon(li[n - 3:n], Color.Blue)
            C.drawLine(zA, zB, Color.Red)
            zA = zB

    C.setXminXmaxYminYmax(-0.1, 1.1, -0.1, 1.1)
    fig = [zB, zA, zC]
    traseaza(fig)
    for k in range(7):
        fig = transforma(fig)
        traseaza(fig)
        if C.mustClose():
            return
    return

if __name__ == '__main__':
    C.initPygame()
    C.run(SierpinskiTriunghiDrepunghic)
```

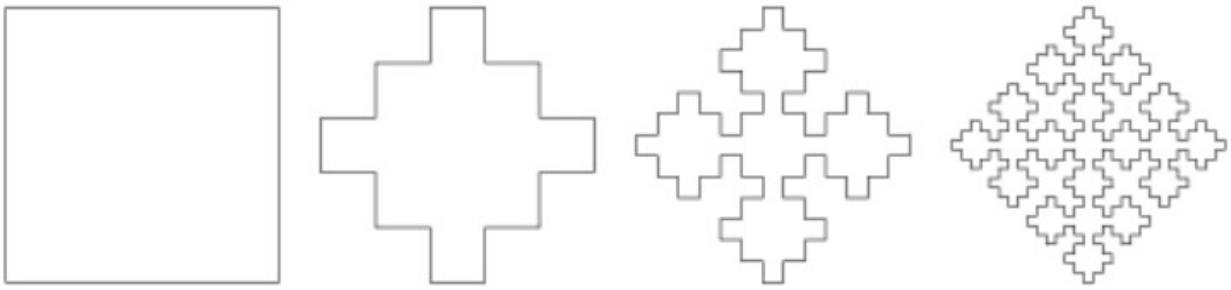
Aici transformarea **s1()** duce triunghiul ΔABC peste triunghiul ΔBDA, iar **s2()** duce pe ΔABC peste ΔADC.

Incercaţi să obţineţi acelaşi rezultat prin metoda motivelor iterate, păstrând la fiecare etapă în lista **fig** triunghiurile obţinute.

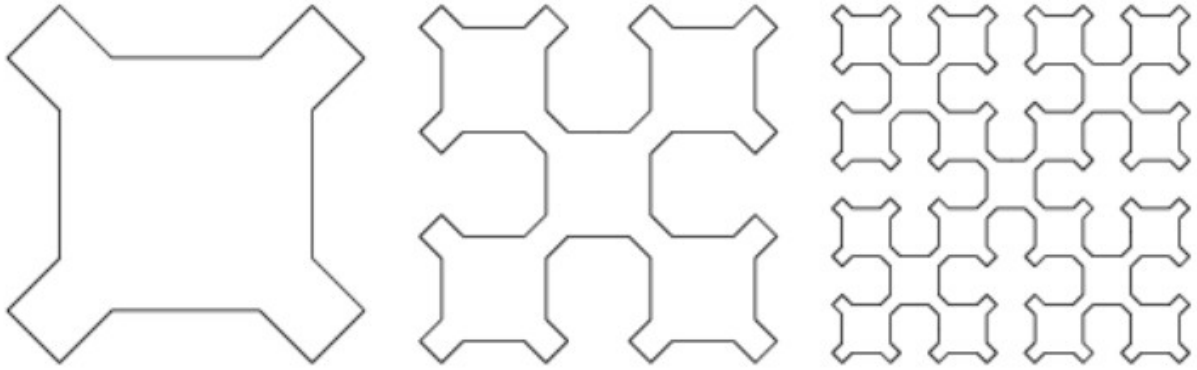


7) Incercaţi să generaţi următoarele c*urbe ale lui Sierpinski*, sugerate de primele etape:

a)



b)



c)