Curs 10
(*plan de curs*)

Curba lui Koch
construcție: *metoda motivelor iterate*.
proprietăţi: *există, nu este rectificabilă, este un arc Jordan, nu are tangentă în nici un punct*.

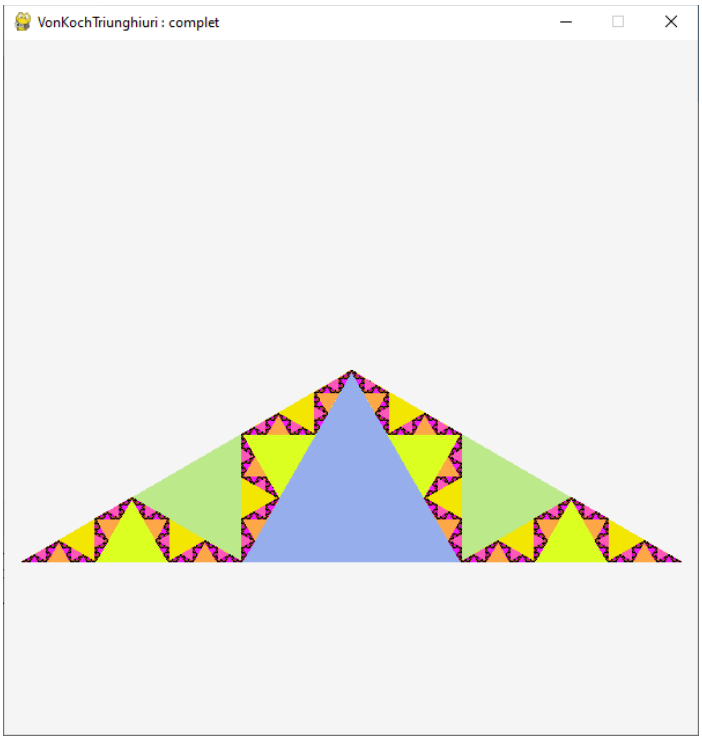
1. Abordare iterativă

1. a) cu liste de numere complexe:

```
def VonKoch():
    theta = math.pi / 6
    rho = 0.5 / math.cos(theta)
    w = C.fromRhoTheta(rho, theta)
    Lambda = 1 / 3.0

    def transforma(li):  # VonKoch
        rez = [li[0]]
        for k in range(1, len(li)):
            z1 = li[k - 1]
            z2 = li[k]
            delta = z2 - z1
            # Segmentul z1_z2 este inlocuit cu
            # z1_zA, zA_zB, zB_zC si zC_z2, unde
            # zA=z1 + Lambda* (z2 - z1)
            # zB=z1 + w * (z2 - z1))
            # zC=z1 + (1-Lambda) * (z2 - z1).
            rez.append(z1 + Lambda * delta)
            rez.append(z1 + w * delta)
            rez.append(z2 - Lambda * delta)
            rez.append(z2)
        return rez

    def traseaza(li):
        C.fillScreen()
        for k in range(1, len(li)):
            C.drawLine(li[k - 1], li[k], Color.Black)
        C.refreshScreen()
        C.wait(50)

    C.setXminXmaxYminYmax(-1.1, 1.1, -0.5, 1.5)
    # segmentul initial
    fig = [-1, 1]
    for k in range(6):
        fig = transforma(fig)
        traseaza(fig)
        if C.mustClose():
            return
```

1.b) cu liste de triunghiuri:

```
def VonKochTriunghiuri():
    theta = math.pi / 6
    rho = 0.5 / math.cos(theta)
    w = C.fromRhoTheta(rho, theta)
    Lambda = 1 / 3.0

    class Triunghi:
        def __init__(self, a, b, c):
            self.a = a
            self.b = b
            self.c = c

        def show(self, col):
            C.drawLine(self.a,self.b,col)
            C.drawLine(self.c, self.b, col)

        def fill(self, col):
            C.fillNgon([self.a, self.b, self.c], col)
            C.drawNgon([self.a, self.b, self.c], col)

    def transforma(li):  # VonKoch
        rez = []
        for t in li:
            z1 = t.a
            zB = t.b
            z2 = t.c
            zA = z1 + Lambda * (z2 - z1)
            zC = z1 + (1 - Lambda) * (z2 - z1)
            rez.append(Triunghi(z1, zA, zB))
            rez.append(Triunghi(zB, zC, z2))
        return rez

    def traseaza(li, col):
        #C.fillScreen()
        for t in li:
            t.fill(col)
            t.show(Color.Black)

    C.setXminXmaxYminYmax(-2, 2, -1, 3)
    # triunghiul initial
    z1 = -1.9
    z2 = +1.9
    zB = z1 + w * (z2 - z1)
    fig = [Triunghi(z1, zB, z2)]
    kmax = 10
    for k in range(kmax):
        traseaza(fig, Color.Index(50 * k + 100))
        fig = transforma(fig)
        if C.mustClose():
            return
        C.wait(50)
```



2. Abordare recursivă:

2.a) cu trasare directă:

```
def VonKochRecursiv():
    theta = math.pi / 6
    rho = 0.5 / math.cos(theta)
    w = C.fromRhoTheta(rho, theta)
    Lambda = 1 / 3.0

    def aplicaMotiv(z1, z2, nivel):
        if nivel <= 0:
            # trasam numai curba finala
            C.drawLine(z1, z2, Color.Navy)
            return

        # zA = z1 + Lambda * (z2 - z1)
        # zB = z1 + w * (z2 - z1)
        # zC = z1 + (1 - Lambda) * (z2 - z1)
        nivel -= 1
        aplicaMotiv(z1, z1 + Lambda * (z2 - z1), nivel)
        aplicaMotiv(z1 + Lambda * (z2 - z1), z1 + w * (z2 - z1), nivel)
        aplicaMotiv(z1 + w * (z2 - z1), z1 + (1 - Lambda) * (z2 - z1), nivel)
        aplicaMotiv(z1 + (1 - Lambda) * (z2 - z1), z2, nivel)

    C.setXminXmaxYminYmax(-1.1, 1.1, -0.5, 1.5)
    aplicaMotiv(-1, 1, 7)
    C.refreshScreen()
```

2.b) cu formarea listei in mod recursiv:

```
def VonKochRecursivList():
    theta = math.pi / 6
    rho = 0.5 / math.cos(theta)
    w = C.fromRhoTheta(rho, theta)
    Lambda = 1 / 3.0
    li = []

    def aplicaMotiv(z1, z2, nivel):
        # nu trimitem lista prin referinta
        # pentru ca nu o schimbam ci numai o completam,
        # o singura data, pe nivelul 0
        if nivel <= 0:
            li.append(z2)
            return

        zA = z1 + Lambda * (z2 - z1)
        zB = z1 + w * (z2 - z1)
        zC = z1 + (1 - Lambda) * (z2 - z1)
        nivel -= 1
        aplicaMotiv(z1, zA, nivel)
        aplicaMotiv(zA, zB, nivel)
        aplicaMotiv(zB, zC, nivel)
        aplicaMotiv(zC, z2, nivel)

    def traseaza(li):

        C.fillScreen()
        for k in range(1, len(li)):
            C.drawLine(li[k - 1], li[k], Color.Navy)
        #  C.wait(0.5)

    C.setXminXmaxYminYmax(-1.1, 1.1, -0.5, 1.5)
    z1 = -1
    z2 = 1
    li.append(z1)
    aplicaMotiv(z1, z2, 5)
    traseaza(li)
    C.refreshScreen()
```