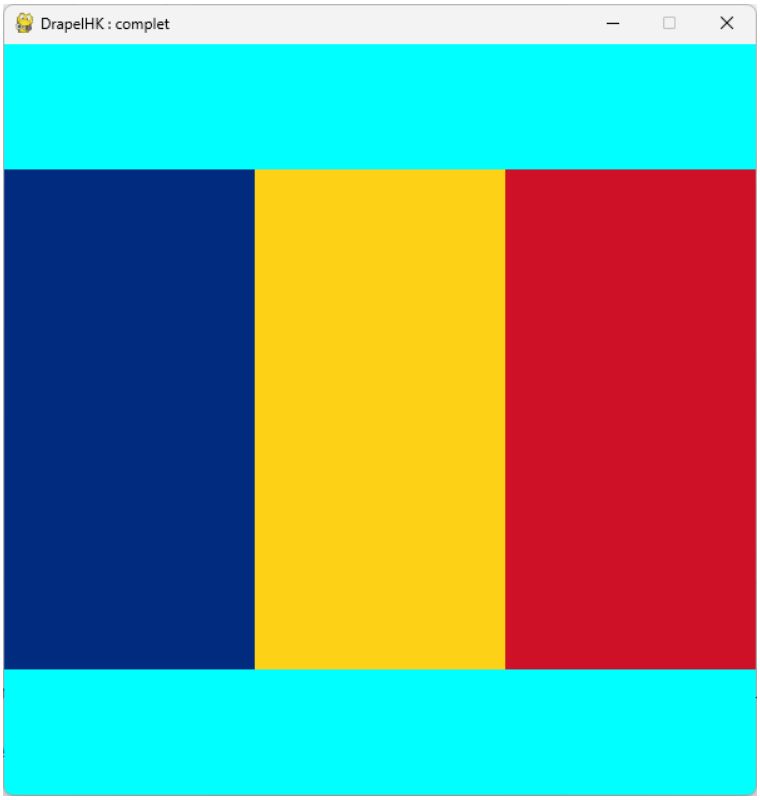
Desenarea pixel cu pixel

Pentru a colora pixel cu pixel întreg bitmapul pus la dispoziția noastră de metoda `initPygame()`, putem folosi funcția `setPixelHK()`, ținând cont că bitmapul are `dim x dim` pixeli. Notăm cu `(h,k)` coordonatele acestor pixeli, `h` pe orizontală, de la stânga la dreapta, și `k` pe verticală, de jos în sus.

Exemplul 1. Următorul program colorează drapelul nostru național[1]:




```
import ComplexPygame as C
import Color

def DrapelHK():
    C.fillScreen()
    for h in range(C.dim):
        for k in range(C.dim):
            col = Color.Aqua
            if C.dim // 6 < k < 5 * C.dim // 6:
                if h < C.dim // 3:
                    col = 0, 43, 127  # albastru cobalt
                elif h < 2 * C.dim // 3:
                    col = 252, 209, 22  # galben crom
                else:
                    col = 206, 17, 38  # rosu vermillon
            C.setPixelHK(h, k, col)
        C.refreshScreen()
    print("gata")

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

1 vezi Drapelul Romaniei

Pentru lucrul cu pixeli `(h,k)` modulul `ComplexPygame.py` pune la dispoziție următoarele metode:

```
def getZ(h, k):
    # returneaza afixul z al punctului corespunzator pixelului (h,k) din ecran
    # (h, k) = (0, 0) este coltul din STANGA JOS
    return complex(xmin + h * dxdh, ymin + k * dydk)

def getXY(h, k):
    # returneaza punctul (x,y) corespunzator pixelului (h,k) din ecran
    return xmin + h * dxdh, ymin + k * dydk

def getHK(z):
    # returneaza pixelul  (h,k) corespunzator lui z complex
    return round((z.real - xmin) * dhdx), round((z.imag - ymin) * dkdy)

def setPixelHK(h, k, color):
    # seteaza pe ecran pixelul de coordonate (h,k)
    screen.set_at((h, dim - k), color)

def drawLineHK(h0, k0, h1, k1, color):
    # traseaza segmentul (h0,k0) (h1,k1) pe ecran
    pygame.draw.line(screen, color, (h0, dim - k0), (h1, dim - k1))
```

Acestea sunt utile, dar lucrul cu coordonate întregi este laborios și, de cele mai multe ori, poate fi evitat folosind *lista numerelor complexe reprezentabile în bitmap*, listă returnată de metoda

```
def screenAffixes():
    # returneaza lista celor dim*dim numere complexe reprezentabile pe ecran
    return [getZ(h, k) for h in range(dim) for k in range(dim)]
```

Iată varianta cu `setPixel()` a funcției de mai sus

```
def DrapelZ():
    lat = 12
    C.setXminXmaxYminYmax(0, lat, 0, lat)
    C.fillScreen()
    for z in C.screenAffixes():
        col = Color.Aqua
        if lat // 6 < z.imag < 5 * lat // 6:
            if z.real < lat // 3:
                col = Color.Blue
            elif z.real < 2 * lat // 3:
                col = Color.Yellow
            else:
                col = Color.Red
        C.setPixel(z, col)
    C.refreshScreen()
    print("gata")
```

Între aceste două variante există o mică diferență: în primul cazul apelul `C.refreshScreen()` este situat în interiorul `for`-ului după `h`, deci imaginea este reîmprospătată după fiecare coloană setată în memorie, pe când în al doilea caz imaginea este reîmprospătată doar la final.

În acest exemplu lucrurile se desfășoară rapid deoarece sunt foarte puține calcule de făcut și imaginea finală apare imediat, dar dacă vrem să vedem și "rezultate intermediare", fără să reîmprospătatăm bitmapul chiar la fiecare pixel setat, avem la dispoziție metoda

```
def screenColumns():
    # genereaza dim liste cu numerele complexe reprezentabile
    # asezate pe coloane
    for h in range(dim):
        yield [getZ(h, k) for k in range(dim)]
```

După cum se observă `screenColumns()` este o *funcție generatoare*[2] (are `yield` în loc de `return`), prin urmare la fiecare apel îl incrementează pe `h` și returnează apoi coloana corespunzătoare acestuia:

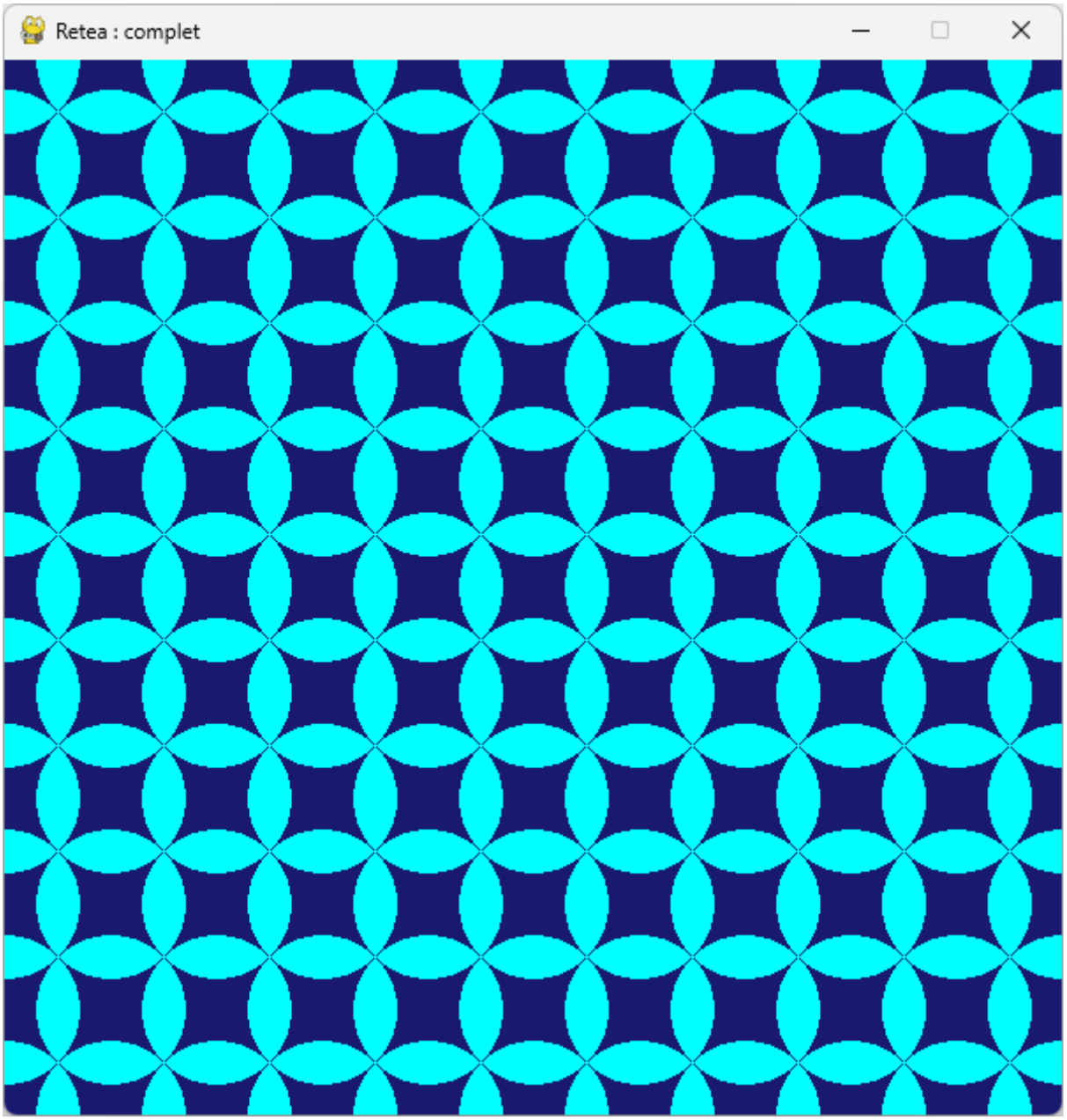
```
def DrapelZcoloane():
    lat = 12
    C.setXminXmaxYminYmax(0, lat, 0, lat)
    C.fillScreen()
    for coloana in C.screenColumns():
        for z in coloana:
            col = Color.Aqua
            if lat // 6 < z.imag < 5 * lat // 6:
                if z.real < lat // 3:
                    col = Color.Blue
                elif z.real < 2 * lat // 3:
                    col = Color.Yellow
                else:
                    col = Color.Red
            C.setPixel(z, col)
        C.refreshScreen()
    print("gata")
```

Exemplul 2. Următoarea funcție de desenare colorează o rețea de cercuri care se suprapun parțial:

```
def Retea():
    N = 10
    C.setXminXmaxYminYmax(0, N, 0, N)
    C.fillScreen()
    centre = [complex(x, y) for x in range(N + 1) for y in range(N + 1)]
    raza = 1 / math.sqrt(2)
    for coloana in C.screenColumns():
        for z in coloana:
            niv = 0
            for c in centre:
                if abs(z - c) < raza:
                    niv += 1
            col = Color.Midnightblue if niv == 1 else Color.Aqua
            C.setPixel(z, col)
        C.refreshScreen()
    print("gata")
```

Puneți în comentriu apelul `C.refreshScreen()` ca să vedeți după cât timp apare imaginea finală.

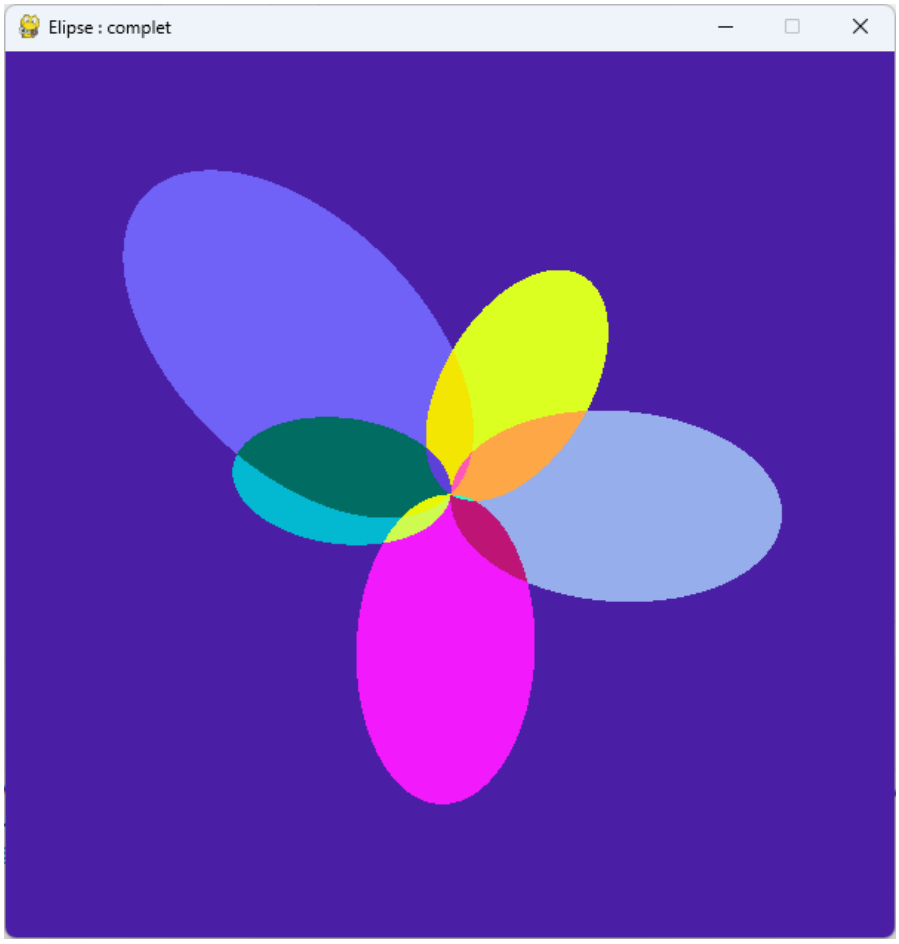
2 vezi, de exemplu, https://realpython.com/introduction-to-python-generators/



Exemplul 3. Ne propunem să colorăm distinct regiunile determinate de cinci elipse care trec printr-un punct U dat. Este binecunoscut faptul că, fiind date trei puncte distincte din plan, F_1, F_2 şi U, există o singură elipsă care trece prin U şi are focarele F_1 şi F_2. Un punct Z este interior acestei elipse dacă suma distanţelor ZF_1 şi ZF_2 este mai mică decât suma distanţelor UF_1 şi UF_2 (sumă notată cu `doi_a` în programul următor, vezi şi Exerciţiul 7 din Tema 3).

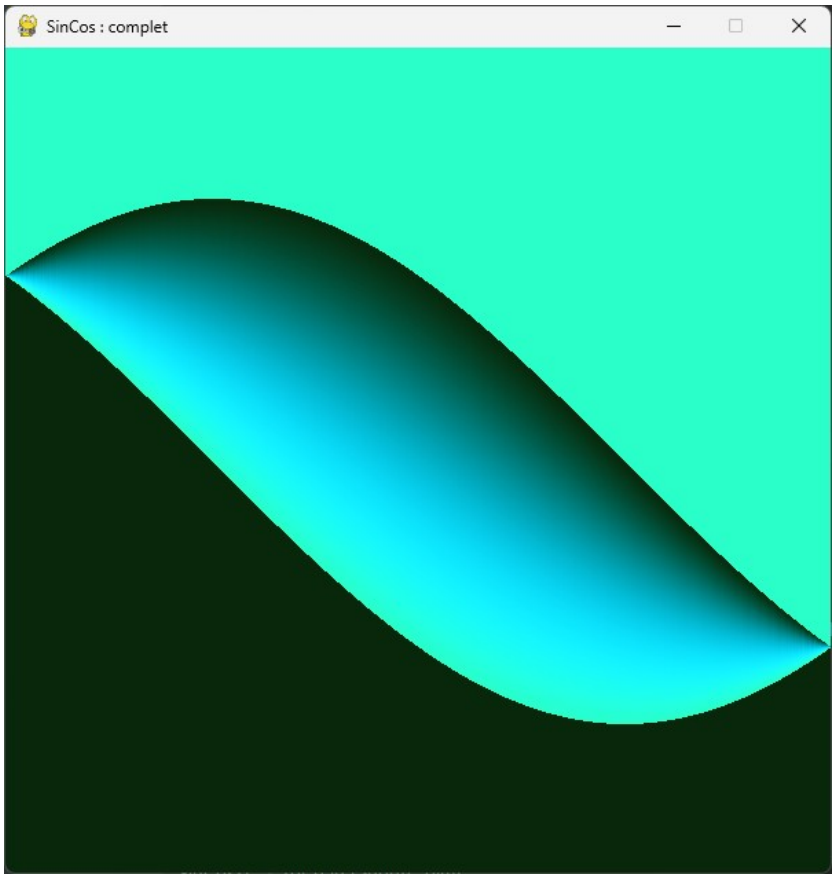
In program punctul comun U este fixat în centrul ecranului iar pentru focarele celor N=5 elipse folosim metoda `random.random()` care returnează numere aleatoare uniform distribuite în intervalul [0, 1]. Pentru fiecare elipsă focarul F_1 este fixat aleator iar F_2 este ales pe segmentul UF_1 într-un raport fix, 9/10, astfel că aceste elipse sunt *asemenea* între ele.

```
import ComplexPygame as C
import Color
import random

def Elipse():
    def esteInElipsa(z, elipsa):
        f1, f2, doi_a = elipsa
        return abs(z - f1) + abs(z - f2) < doi_a

    C.setXminXmaxYminYmax(-0.2, 1.2, -0.2, 1.2)
    u = 0.5 + 0.5j
    elipse = []
    N = 5
    for k in range(N):
        f1 = complex(random.random(), random.random())
        f2 = (f1 + 9 * u) / 10
        doi_a = abs(u - f1) + abs(u - f2)
        elipse.append((f1, f2, doi_a))
    for z in C.screenAffixes():
        niv = 0
        for k, elipsa in enumerate(elipse):
            if esteInElipsa(z, elipsa):
                niv += 2 ** k
        C.setPixel(z, Color.Index(50 * niv))
    C.refreshScreen()

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



Exemplul 4. Următoarea funcție de desenare umple cu un gradient de culoare regiunea cuprinsă între graficele funcțiilor *sinus* și *cosinus* între două puncte de intersecție consecutive.

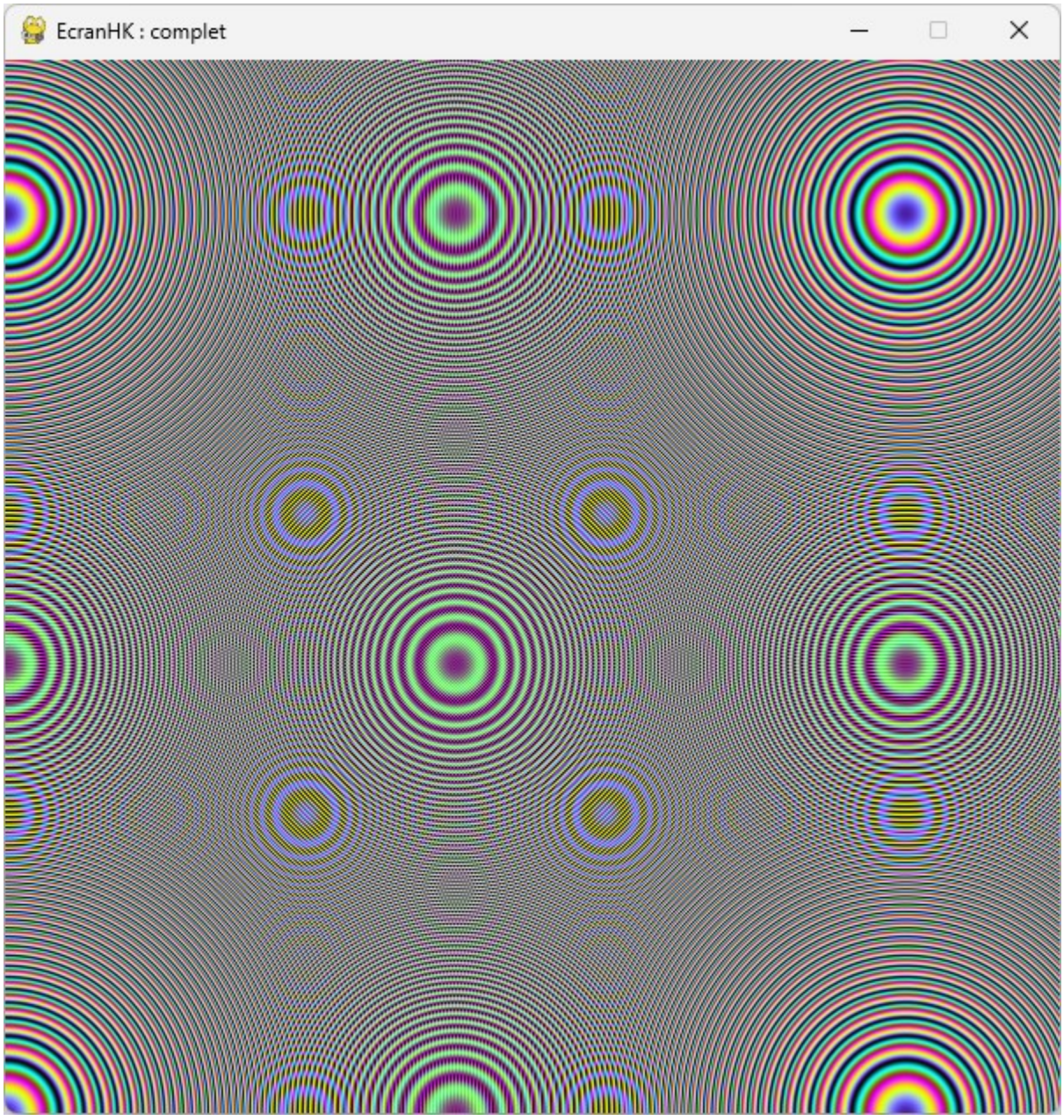
```
def SinCos():
    pi4 = math.pi / 4
    C.setXminXmaxYminYmax(pi4, 5 * pi4, -2 * pi4, 2 * pi4)
    kol_cos = 700
    delta_kol = 200
    col_cos = Color.Index(kol_cos)
    col_sin = Color.Index(kol_cos + delta_kol)
    for h in range(C.dim):
        x, _ = C.getXY(h, 0)
        zsin = complex(x, math.sin(x))
        zcos = complex(x, math.cos(x))
        _, ksin = C.getHK(zsin)
        _, kcos = C.getHK(zcos)
        deltak = ksin - kcos
        C.drawLineHK(h, 0, h, kcos, col_sin)
        for k in range(kcos, ksin):
            kol = round(kol_cos + delta_kol * (k - kcos) / deltak)
            C.setPixelHK(h, k, Color.Index(kol))
        C.drawLineHK(h, ksin, h, C.dim, col_cos)
    C.refreshScreen()
```



După cum se vede, bitmapul este colorat coloană cu coloană, pentru fiecare index de coloană *h* determinăm *x*-ul corespunzător lui în plan, apoi aflăm indexurile *k* corespunzătoare lui sin(*x*) și cos(*x*), după care trasăm o linie verticală de jos până la (*h*, *kcos*), umplem pixel cu pixel segmentul de la (*h*, *kcos*) la (*h*, *ksin*) și în final terminăm coloana cu o altă linie verticală, de la (*h*, *ksin*) până sus.

In final, o **întrebare**: cum se explică apariţia reţelelor de cercuri concentrice în imaginea generată de programul următor?

```
import ComplexPygame as C
import Color
def EcranHK():
    C.fillScreen()
    for h in range(C.dim):
        for k in range(C.dim):
            C.setPixelHK(h, k, Color.Index(h * h + k * k))
    C.refreshScreen()

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



Indicaţie: vedeţi ce se întâmplă dacă schimbaţi modul de colorare astfel:

```
            C.setPixelHK(h, k, Color.Index((h * h + k * k) % 200))
```