Cursul 2

Clase Python pentru grafică 2D

1. Pachetul Pygame

Pentru a realiza o aplicație de grafică animată avem nevoie de un bitmap (o matrice de pixeli), de o metodă de setare a fiecărui pixel și de o posibilitate de lansare/oprire a animației după dorință. Toate acestea ne sunt puse la dispoziție de biblioteca Pygame, un proiect indexat de `PyPI` la adresa `https://pypi.org/project/pygame/` și care, prin urmare, poate fi instalat cu comanda shell

```
pip install pygame
```

Metodele esențiale ale pachetului Pygame sunt, în câteva cuvinte, următoarele:

`pygame.init()`

inițializează toate modulele pachetului și pornește bucla de tratare a evenimentelor,

`pygame.display.set_mode(size)`

returnează o "suprafață", o instanță a clasei Surface, formată dintr-o fereastră vizibilă pe ecran dotată cu un bitmap,

`pygame.Surface.set_at ((x, y), Color) )`

setează în bitmapul suprafeței culoarea pixelului de coordonate (x,y),

`pygame.display.flip()`

face vizibil desenul din bitmap (se utilizează tehnica *double buffering)* și

`pygame.event.get()`

scoate din coada de așteptare toate mesajele Windows care privesc aplicația noastră și ni le furnizează pentru a fi tratate.

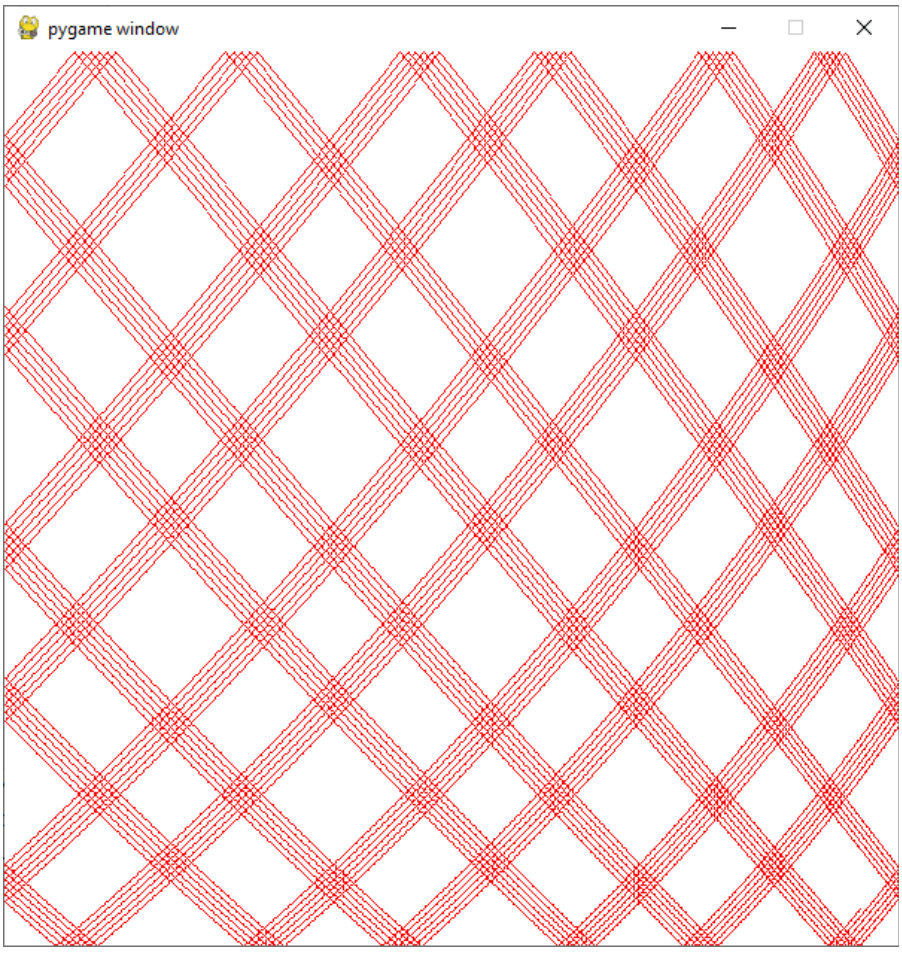
Exemplul minimal de utilizare este dat de următorul program în care un punct de culoare roșie se mișcă la nesfârșit pe un fond alb, până când utilizatorul închide fereastra aplicației prin clic cu mausul în colțul ei din dreapta sus:

```
import pygame
import math
def Exemplul_1():
    def minsincos(s):
        return min(abs(math.sin(s)), abs(math.cos(s)))

    white = 255, 255, 255
    red = 255, 0, 0
    dim = 600
    ddim = dim * math.sqrt(2.0)
    screen = pygame.display.set_mode((dim, dim))
```

```
    screen.fill(white)
    t = 0
    isRunning = True
    while isRunning:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                isRunning = False
        x = round(ddim * minsincos(t))
        y = round(ddim * minsincos(1.201 * t + 0.2024))
        screen.set_at((x, y), red)
        pygame.display.flip()
        t += 0.001
    print("Gata!")

if __name__ == '__main__':
    pygame.init()
    Exemplul_1()
```



2. Modulul ComplexPygame.py

Modulul ComplexPygame descris mai jos a fost implementat special pentru acest curs și cuprinde metodele strict necesare pentru dezvoltarea de aplicaţii Python de grafică animată 2D cu ajutorul numerelor complexe.

Fișierul ComplexPygame.py începe cu declararea și inițializarea unor variabile globale:

```
import pygame
import Color
import cmath

# variabile globale:
dim = 600
xmin = 0.0
xmax = 1.0
ymin = 0.0
ymax = 1.0
dxdh = 1.0 / dim
dydk = 1.0 / dim
dhdx = float(dim)
dkdy = float(dim)
screen = pygame.Surface((dim, dim))
mustExit = True
mustPainting = False
```

Se observă că avem la dispoziție, în mod implicit, un bitmap cu `600 X 600` de pixeli în care trebuie să reprezentăm un desen din planul *xOy*, încadrat într-un dreptunghi `[xmin, xmax] X [ymin, ymax]`, setat de utilizator prin apelul funcției

```
def setXminXmaxYminYmax(xxmin=0, xxmax=1, yymin=0, yymax=1):
    global xmin, xmax, ymin, ymax, dxdh, dydk, dhdx, dkdy
    xmin = xxmin
    xmax = xxmax
    ymin = yymin
    ymax = yymax
    dxdh = (xmax - xmin) / dim
    dydk = (ymax - ymin) / dim
    dhdx = dim / (xmax - xmin)
    dkdy = dim / (ymax - ymin)
```

În orice program de grafică bi-dimensională trebuie să utilizăm două sisteme de referinţă: cel al punctelor din plan, de coordonate (*x,y*), cu *x* şi *y* numere reale, şi cel al pixelilor de pe ecran, de coordonate (*h, k*), cu *h* şi *k* numere întregi. Pe ecran apar numai pixelii din dreptunghiul delimitat de `hmin, hmax, kmin, kmax,` care conţine imaginea din planul *xOy* aflată în dreptunghiul delimitat de `xmin, xmax, ymin, ymax.`

Numărul complex $z=x+iy$, afixul punctului $z=(x,y)$, este reprezentabil numai dacă x este între `xmin` și `xmax`, iar y între `ymin` și `ymax`. Să notam, provizoriu, `hmin =0`, `hmax = dim`, `kmin = 0` și `kmax = dim`.

Transformările afine care suprapun cele două dreptunghiuri folosesc factorii de scalare `dxdh, dydk, dhdx, dkdy,` calculaţi astfel:

```
dxdh = (xmax - xmin) / (hmax - hmin);
dydk = (ymax - ymin) / (kmax - kmin);
dhdx = (hmax - hmin) / (xmax - xmin);
djdy = (kmax - kmin) / (ymax - ymin);
```

Trecerea de la coordonatele (x, y) la (h, k) este dată de formulele:

```
h = round(hmin + (x - xmin) * dhdx)
k = round(kmin + (y - ymin) * dkdy)
```

iar trecerea inversă de formulele:

```
x = xmin+(h-hmin)*dxdi
y = ymin+(k-hmin)*dydk
```

Aceste formule sunt folosite în următoarele funcții:

```
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)
```

Pentru a seta (a colora) un pixel trebuie să folosim metoda Pygame `screen.set_at((i, j), color)` care completează bitmapul de sus în jos, ca în scrierea în consolă, pentru a obține desenul în poziția obisnuită, cu pixelul `(0,0)` în colțul din dreapta jos, la apelul metodelor Pygame trebuie să-l trecem pe `k` in `dim-k`. Obținem pentru desenare următoarele funcții:

```
def setPixelXY(x, y, color):
    # seteaza pixelul corespunzator punctului  (x,y) din plan
    screen.set_at((round((x - xmin) * dhdx), round((ymax - y) * dkdy)), color)


def setPixel(z, color):
    # seteaza pixelul corespunzator numarului complex z
```

```
    screen.set_at((round((z.real - xmin) * dhdx), round((ymax - z.imag) * dkdy)),
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))

def drawLineXY(x0, y0, x1, y1, color):
    # traseaza segmentul (x0,y0) (x1,y1) din plan
    pygame.draw.line(screen, color, (round((x0 - xmin) * dhdx), round((ymax - y0) *
dkdy)),
                     (round((x1 - xmin) * dhdx), round((ymax - y1) * dkdy)))

def drawLine(z0, z1, color):
    # traseaza segmentul z0 z1 din planul complex
    pygame.draw.line(screen, color, (round((z0.real - xmin) * dhdx), round((ymax -
z0.imag) * dkdy)),
                     (round((z1.real - xmin) * dhdx), round((ymax - z1.imag) *
dkdy)))

def setNgon(z_list, color):
    # seteaza varfurile poligonului z_list = [z0, z1, ...]
    for z in z_list:
        setPixel(z, color)
    return

def drawNgon(z_list, color):
    # traseaza laturile poligonului z_list = [z0, z1, ...]
    pygame.draw.polygon(screen, color, [_getIJ(z) for z in z_list], width=1)

def fillNgon(z_list, color):
    # umple poligonul convex z_list = [z0, z1, ...]
    pygame.draw.polygon(screen, color, [_getIJ(z) for z in z_list])
```

Pentru desenare mai avem nevoie de funcțiile

```
def fillScreen(color=Color.Whitesmoke):
    screen.fill(color)

def refreshScreen():
    pygame.display.flip()
```

care, așa cum le arată numele, stabilesc culoarea fundalului și reîmprospătează imaginea pe ecran. Detalii privind metodele Pygame de desenare se găsesc la adresa web:

https://www.pygame.org/docs/ref/draw.html

Acum putem desena, mai trebuie doar să lansăm desenarea și, în cazul unei animații de prea mare durată, să o oprim. Pentru lansare vom folosi funcția următoare:

```
def run(drawing_function):
    # lanseaza in executie functia care realizeaza desenul
    print(f"start run({drawing_function.__name__})")
    fillScreen()
    pygame.display.set_caption(drawing_function.__name__ + " : apasati bara de spatiu
")
    pygame.display.flip()
    global mustExit, mustPainting
    while True:  # main loop
        mustExit = False
        mustPainting = False
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                mustExit = True
            if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
                mustPainting = True
        if mustExit:
            break  # main loop
        if mustPainting:
            pygame.display.set_caption(drawing_function.__name__ + " : in lucru")
            # lansam desenarea:
            drawing_function()
            # dupa terminarea desenarii:
            pygame.display.flip()
            if mustExit:
                break  # main loop
            if mustPainting:
                pygame.display.set_caption(drawing_function.__name__ + " : complet")
            else:
                pygame.display.set_caption(drawing_function.__name__ + " : oprit")
    print(f"exit run({drawing_function.__name__})")
```

Aici, `drawing_function` este funcția care realizează desenul. In timpul execuției ei desenarea poate fi oprită cu

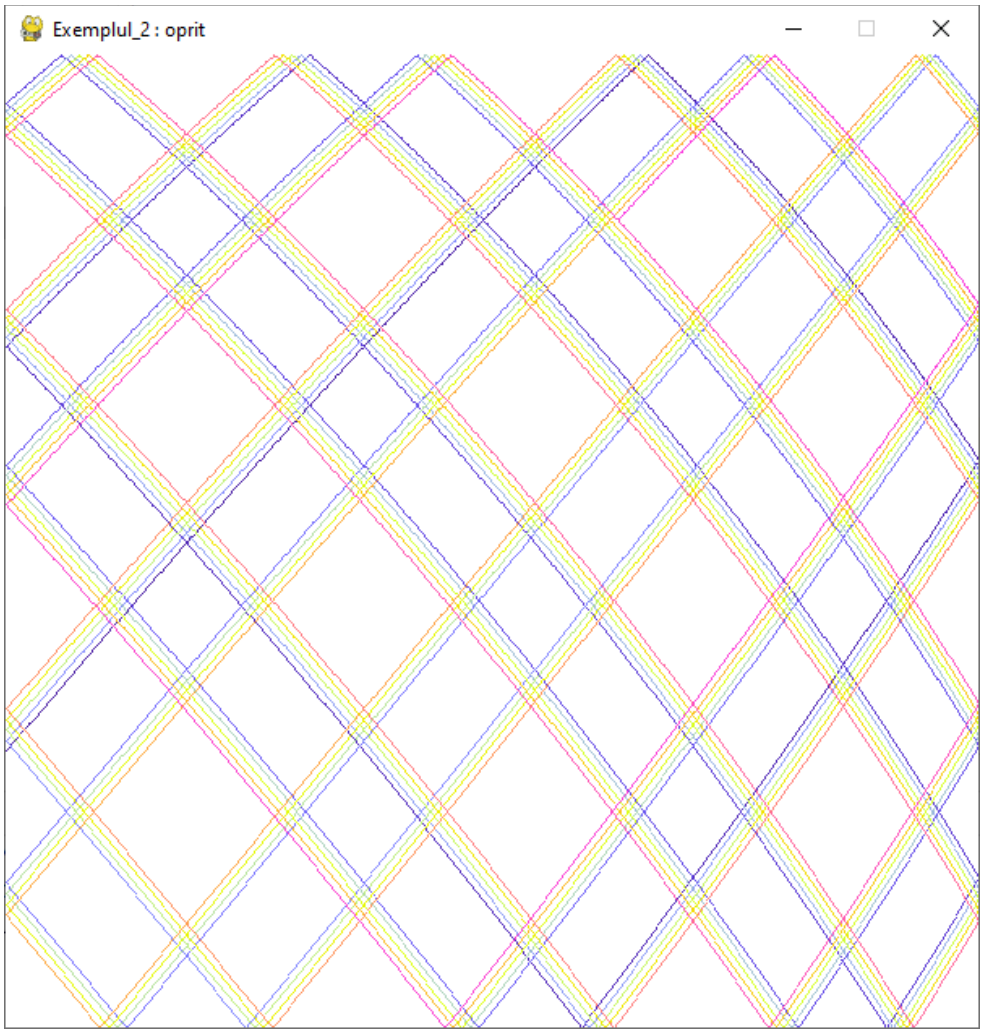
```
def mustClose():
    # poate fi apelata in timpul desenarii, pentru oprirea desenarii
    pygame.display.flip()
    global mustExit
    global mustPainting
    mustStop = False
    for event in pygame.event.get():
```

```
        if event.type == pygame.QUIT:
            mustExit = True
        if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
            mustStop = True
            mustPainting = False
    return mustExit or mustStop
```

Următorul program realizează același desen ca în exemplul precedent, cu deosebirea esențială că acum desenarea este lansată și oprită cu bara de spațiu:

```
import ComplexPygame as C
import Color
import math, sys
def Exemplul_2():
    def minsincos(s):
        return min(abs(math.sin(s)), abs(math.cos(s)))
    lat=10
    C.setXminXmaxYminYmax(0, lat, 0, lat)
    ddim = lat * math.sqrt(2.0)
    C.fillScreen(Color.White)
    for k in range(sys.maxsize):
        t = 0.001 * k
        x = ddim * minsincos(t)
        y = ddim * minsincos(1.201 * t + 0.2024)
        C.setPixelXY(x,y,Color.Index(k // 100))
        if  C.mustClose():
            break
    print("Gata!")

if __name__ == '__main__':
    C.initPygame()
    C.run(Exemplul_2)
```



3. Fisierul Color.py

Modul Color pune la dispoziția utilizatorului o paletă cu 1024 culori furnizată de funcția `Color.Index(k)` și o serie de nume de culori definite prin constante de tip string hexazecimal RGB:

```
import math

_color = []
for k in range(1024):
    tcol = 56.123 + 2.0 * math.pi * k / 1024
    rcol = int(128 + 128 * math.sin(tcol))
    gcol = int(128 + 128 * math.sin(2 * tcol))
    bcol = int(128 + 128 * math.cos(3 * tcol))
    _color.append((rcol % 256, gcol % 256, bcol % 256,))

def Index(k):
    return _color[k % 1024]
```

```
# https://stackoverflow.com/questions/22408237/named-colors-in-matplotlib

Aliceblue = '#F0F8FF'
Antiquewhite = '#FAEBD7'
Aqua = '#00FFFF'
Aquamarine = '#7FFFD4'
Azure = '#F0FFFF'
Beige = '#F5F5DC'
Bisque = '#FFE4C4'
Black = '#000000'
................................................
```

Utilizarea lor a fost deja exemplificată în programul precedent.

4. Fișierul PythonSablon.py

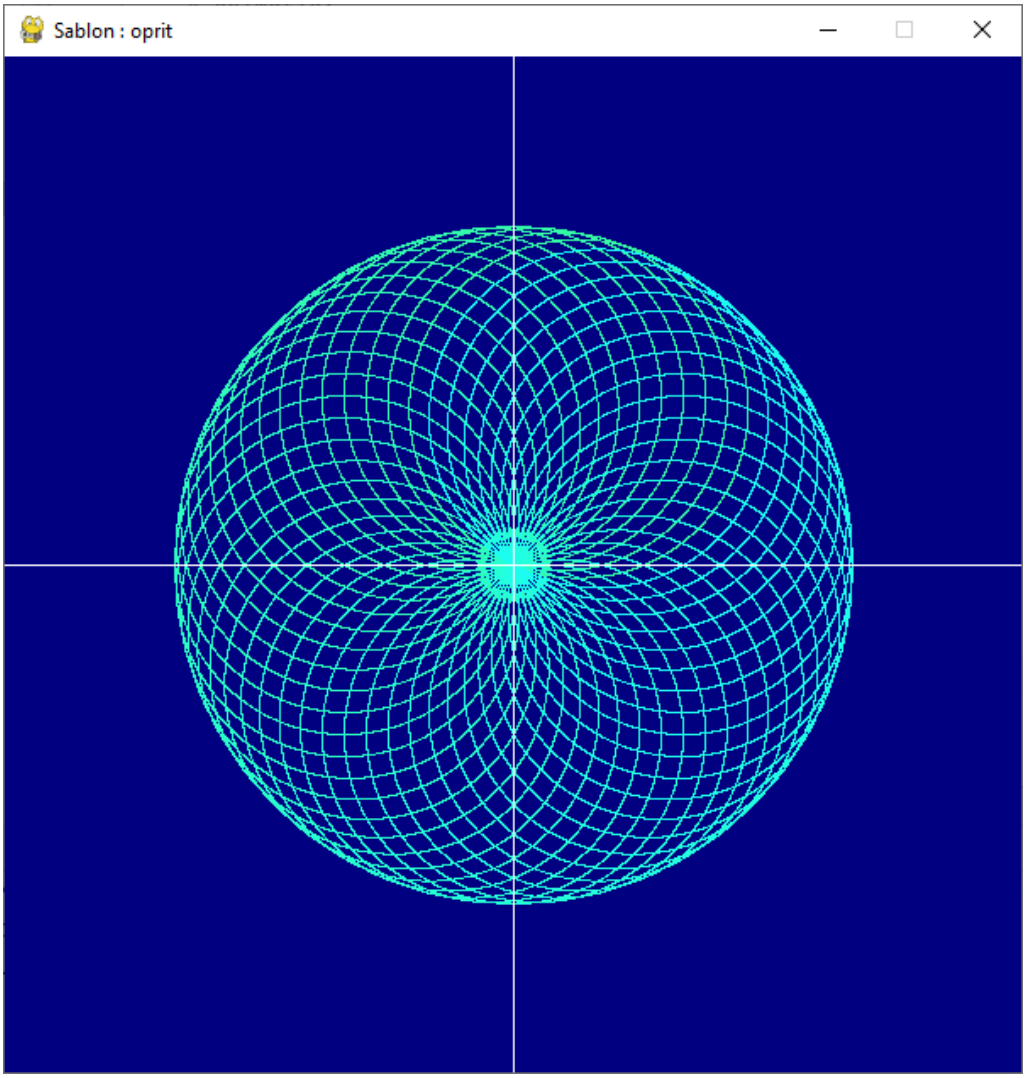
Pentru exemplificarea utilizării numerelor complexe prezentăm proiectul Șablon:

```
import ComplexPygame as C
import Color
import math
import sys

def Sablon():
    def FiguraInitiala(r):
        # un cerc care trece prin origine
        nrPuncte = 5000
        delta = 2 * math.pi / nrPuncte
        return [r + C.fromRhoTheta(r, h * delta) for h in range(nrPuncte)]

    # animatia:
    C.fillScreen(Color.Navy)
    lat = 1
    C.setXminXmaxYminYmax(-lat, lat, -lat, lat)
    fig = FiguraInitiala(lat / 3)
    omega = C.fromRhoTheta(1, math.pi / 24)
    for t in range(sys.maxsize):
        # C.fillScreen(Color.Navy)
        col = Color.Index(t)
        for k in range(len(fig)):
            C.setPixel(fig[k], col)
            fig[k] *= omega
        if C.mustClose():
            break
    C.setAxis(Color.White)
```

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



Aici, la început în variabila `fig` este încărcată o listă de 5000 de numere complexe dispuse uniform pe un cerc de rază `r` cu centrul în punctul de coordonate `(r,0)` și apoi, în ciclul `for` după indexul `t`, la fiecare etapă este desenată punct cu punct întreaga figură memorată în listă și apoi, prin amplificare cu numărul complex `omega` de modul unu, figura este rotită cu un unghi de `math.pi/24` radiani în jurul originii.

Anexe

Fișierul ComplexPygame.py

```
# versiune: 26.01.2024
import pygame
import Color
import cmath

# variabile globale:
dim = 600
xmin = 0.0
xmax = 1.0
ymin = 0.0
ymax = 1.0
dxdh = 1.0 / dim
dydk = 1.0 / dim
dhdx = float(dim)
dkdy = float(dim)
screen = pygame.Surface((dim, dim))
mustExit = True
mustPainting = False

# redenumiri de functii:
rho = abs
theta = cmath.phase
fromRhoTheta = cmath.rect
wait = pygame.time.wait


def initPygame(dm=600):
    global screen, dim
    dim = dm
    print("start pygame")
    pygame.init()
    screen = pygame.display.set_mode((dim, dim))


def setXminXmaxYminYmax(xxmin=0.0, xxmax=1.0, yymin=0.0, yymax=1.0):
    global xmin, xmax, ymin, ymax, dxdh, dydk, dhdx, dkdy
    xmin = xxmin
    xmax = xxmax
    ymin = yymin
    ymax = yymax
    dxdh = (xmax - xmin) / dim
    dydk = (ymax - ymin) / dim
    dhdx = dim / (xmax - xmin)
    dkdy = dim / (ymax - ymin)
```

```
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 _getIJ(z):
    # pentru uz privat: (i, j) == (h, dim - k)
    # furnizeaza coordonatele (i,j) din bitmap
    # coordonate folosite de set_at si get_at
    # set_at((0,0),color) este coltul din STANGA SUS
    return round((z.real - xmin) * dhdx), round((ymax - z.imag) * dkdy)


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)]


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)]


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


def setPixelXY(x, y, color):
    # seteaza pixelul corespunzator punctului  (x,y) din plan
    screen.set_at((round((x - xmin) * dhdx), round((ymax - y) * dkdy)), color)
```

```
def setPixel(z, color):
    # seteaza pixelul corespunzator numarului complex z
    screen.set_at((round((z.real - xmin) * dhdx), round((ymax - z.imag) * dkdy)),
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))


def drawLineXY(x0, y0, x1, y1, color):
    # traseaza segmentul (x0,y0) (x1,y1) din plan
    pygame.draw.line(screen, color, (round((x0 - xmin) * dhdx), round((ymax - y0) *
dkdy)),
                     (round((x1 - xmin) * dhdx), round((ymax - y1) * dkdy)))


def drawLine(z0, z1, color):
    # traseaza segmentul z0 z1 din planul complex
    pygame.draw.line(screen, color, (round((z0.real - xmin) * dhdx), round((ymax -
z0.imag) * dkdy)),
                     (round((z1.real - xmin) * dhdx), round((ymax - z1.imag) *
dkdy)))


def setNgon(z_list, color):
    # seteaza varfurile poligonului z_list = [z0, z1, ...]
    for z in z_list:
        setPixel(z, color)
    return


def drawNgon(z_list, color):
    # traseaza laturile poligonului z_list = [z0, z1, ...]
    pygame.draw.polygon(screen, color, [_getIJ(z) for z in z_list], width=1)


def fillNgon(z_list, color):
    # umple poligonul convex z_list = [z0, z1, ...]
    pygame.draw.polygon(screen, color, [_getIJ(z) for z in z_list])


def setAxis(color=Color.Black):
    drawLineXY(xmin, 0, xmax, 0, color)
    drawLineXY(0, ymin, 0, ymax, color)
def setTextIJ(text=" ", i=dim // 2, j=dim // 2, color=Color.Navy, size=16):
    # scrie un text pe ecran, (i, j) = (0, 0) este coltul de STANGA SUS
```

```
    myFont = pygame.font.Font('TimeRomanNormal.ttf', size)
    textImag = myFont.render(text, True, color)
    textRect = textImag.get_rect()
    textRect.bottomleft = (i, j)
    screen.blit(textImag, textRect)

def setText(text="O", z=0j, color=Color.Black, size=16):
    # scrie un text pe ecran in dreptul lui z
    myFont = pygame.font.Font('TimeRomanNormal.ttf', size)
    textImag = myFont.render(text, True, color)
    textRect = textImag.get_rect()
    i, j = _getIJ(z)
    textRect.center = (i, j - 3 * size // 4)
    screen.blit(textImag, textRect)

def saveScreenPNG(filename):
    pygame.image.save(screen, filename + ".png")
def fillScreen(color=Color.Whitesmoke):
    screen.fill(color)
def refreshScreen():
    pygame.display.flip()
def run(drawing_function):
    # lanseaza in executie functia care realizeaza desenul
    print(f"start run({drawing_function.__name__})")
    fillScreen()
    pygame.display.set_caption(drawing_function.__name__ + " : apasati bara de spatiu
")
    pygame.display.flip()
    global mustExit, mustPainting
    while True:  # main loop
        mustExit = False
        mustPainting = False
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                mustExit = True
            if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
                mustPainting = True
        if mustExit:
            break  # main loop
        if mustPainting:
            pygame.display.set_caption(drawing_function.__name__ + " : in lucru")
            # lansam desenarea:
            drawing_function()
            # dupa terminarea desenarii:
            pygame.display.flip()
            if mustExit:
                break  # main loop
            if mustPainting:
```

```
                pygame.display.set_caption(drawing_function.__name__ + " : complet")
            else:
                pygame.display.set_caption(drawing_function.__name__ + " : oprit")
    print(f"exit run({drawing_function.__name__})")
def mustClose():
    # poate fi apelata in timpul desenarii, pentru oprirea desenarii
    pygame.display.flip()
    global mustExit, mustPainting
    mustStop = False
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            mustExit = True
        if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
            mustStop = True
            mustPainting = False
    return mustExit or mustStop


if __name__ == '__main__':
    # exemplu de utilizare:
    def desenare():
        setXminXmaxYminYmax(-1, 1, -1, 1)
        fillScreen(Color.Darkturquoise)
        setTextIJ("Pentru oprire apasati bara de spatiu", 10, 20)
        for k in range(10 ** 6):
            z = fromRhoTheta(0.8, k * 0.0005)
            v = fromRhoTheta(0.2, k * 0.009)
            drawLine(z, v, Color.Index(k // 10))
            if mustClose():
                break
        fillScreen(Color.Turquoise)
        setTextIJ("Pentru repornire apasati tot bara de spatiu", 10, 20)
        refreshScreen()
    initPygame()
    run(desenare)
```

Fișierul Color.py

```
import math

_color = []
for k in range(1024):
    tcol = 56.123 + 2.0 * math.pi * k / 1024
    rcol = int(128 + 128 * math.sin(tcol))
    gcol = int(128 + 128 * math.sin(2 * tcol))
```

```
    bcol = int(128 + 128 * math.cos(3 * tcol))
    _color.append((rcol % 256, gcol % 256, bcol % 256,))

def Index(k):
    return _color[k % 1024]


# https://stackoverflow.com/questions/22408237/named-colors-in-matplotlib

Aliceblue = '#F0F8FF'
Antiquewhite = '#FAEBD7'
Aqua = '#00FFFF'
Aquamarine = '#7FFFD4'
Azure = '#F0FFFF'
Beige = '#F5F5DC'
Bisque = '#FFE4C4'
Black = '#000000'
Blanchedalmond = '#FFEBCD'
Blue = '#0000FF'
Blueviolet = '#8A2BE2'
Brown = '#A52A2A'
Burlywood = '#DEB887'
Cadetblue = '#5F9EA0'
Chartreuse = '#7FFF00'
Chocolate = '#D2691E'
Coral = '#FF7F50'
Cornflowerblue = '#6495ED'
Cornsilk = '#FFF8DC'
Crimson = '#DC143C'
Cyan = '#00FFFF'
Darkblue = '#00008B'
Darkcyan = '#008B8B'
Darkgoldenrod = '#B8860B'
Darkgray = '#A9A9A9'
Darkgreen = '#006400'
Darkkhaki = '#BDB76B'
Darkmagenta = '#8B008B'
Darkolivegreen = '#556B2F'
Darkorange = '#FF8C00'
Darkorchid = '#9932CC'
Darkred = '#8B0000'
Darksalmon = '#E9967A'
Darkseagreen = '#8FBC8F'
Darkslateblue = '#483D8B'
Darkslategray = '#2F4F4F'
Darkturquoise = '#00CED1'
Darkviolet = '#9400D3'
Deeppink = '#FF1493'
```

```
Deepskyblue = '#00BFFF'
Dimgray = '#696969'
Dodgerblue = '#1E90FF'
Firebrick = '#B22222'
Floralwhite = '#FFFAF0'
Forestgreen = '#228B22'
Fuchsia = '#FF00FF'
Gainsboro = '#DCDCDC'
Ghostwhite = '#F8F8FF'
Gold = '#FFD700'
Goldenrod = '#DAA520'
Gray = '#808080'
Green = '#008000'
Greenyellow = '#ADFF2F'
Honeydew = '#F0FFF0'
Hotpink = '#FF69B4'
Indianred = '#CD5C5C'
Indigo = '#4B0082'
Ivory = '#FFFFF0'
Khaki = '#F0E68C'
Lavender = '#E6E6FA'
Lavenderblush = '#FFF0F5'
Lawngreen = '#7CFC00'
Lemonchiffon = '#FFFACD'
Lightblue = '#ADD8E6'
Lightcoral = '#F08080'
Lightcyan = '#E0FFFF'
Lightgreen = '#90EE90'
Lightgray = '#D3D3D3'
Lightpink = '#FFB6C1'
Lightsalmon = '#FFA07A'
Lightseagreen = '#20B2AA'
Lightskyblue = '#87CEFA'
Lightslategray = '#778899'
Lightsteelblue = '#B0C4DE'
Lightyellow = '#FFFFE0'
Lime = '#00FF00'
Limegreen = '#32CD32'
Linen = '#FAF0E6'
Magenta = '#FF00FF'
Maroon = '#800000'
Mediumaquamarine = '#66CDAA'
Mediumblue = '#0000CD'
Mediumorchid = '#BA55D3'
Mediumpurple = '#9370DB'
Mediumseagreen = '#3CB371'
Mediumslateblue = '#7B68EE'
Mediumturquoise = '#48D1CC'
```

```
Mediumvioletred = '#C71585'
Midnightblue = '#191970'
Mintcream = '#F5FFFA'
Mistyrose = '#FFE4E1'
Moccasin = '#FFE4B5'
Navajowhite = '#FFDEAD'
Navy = '#000080'
Oldlace = '#FDF5E6'
Olive = '#808000'
Olivedrab = '#6B8E23'
Orange = '#FFA500'
Orangered = '#FF4500'
Orchid = '#DA70D6'
Palegoldenrod = '#EEE8AA'
Palegreen = '#98FB98'
Paleturquoise = '#AFEEEE'
Palevioletred = '#DB7093'
Papayawhip = '#FFEFD5'
Peachpuff = '#FFDAB9'
Peru = '#CD853F'
Pink = '#FFC0CB'
Plum = '#DDA0DD'
Powderblue = '#B0E0E6'
Purple = '#800080'
Red = '#FF0000'
Rosybrown = '#BC8F8F'
Royalblue = '#4169E1'
Saddlebrown = '#8B4513'
Salmon = '#FA8072'
Sandybrown = '#FAA460'
Seagreen = '#2E8B57'
Seashell = '#FFF5EE'
Sienna = '#A0522D'
Silver = '#C0C0C0'
Skyblue = '#87CEEB'
Slateblue = '#6A5ACD'
Slategray = '#708090'
Snow = '#FFFAFA'
Springgreen = '#00FF7F'
Steelblue = '#4682B4'
Tan = '#D2B48C'
Teal = '#008080'
Thistle = '#D8BFD8'
Tomato = '#FF6347'
Turquoise = '#40E0D0'
Violet = '#EE82EE'
Wheat = '#F5DEB3'
White = '#FFFFFF'
```

```
Whitesmoke = '#F5F5F5'
Yellow = '#FFFF00'
Yellowgreen = '#9ACD32'
```

Fișierul PythonSablon.py

```
import ComplexPygame as C
import Color
import math
import sys


def Sablon():
    def FiguraInitiala(r):
        # un cerc care trece prin origine
        nrPuncte = 5000
        delta = 2 * math.pi / nrPuncte
        return [r + C.fromRhoTheta(r, h * delta) for h in range(nrPuncte)]

    # animatia:
    C.fillScreen(Color.Navy)
    lat = 1
    C.setXminXmaxYminYmax(-lat, lat, -lat, lat)
    fig = FiguraInitiala(lat / 3)
    omega = C.fromRhoTheta(1, math.pi / 24)
    for t in range(sys.maxsize):
        # C.fillScreen(Color.Navy)
        col = Color.Index(t)
        for k in range(len(fig)):
            C.setPixel(fig[k], col)
            fig[k] *= omega
        if C.mustClose():
            break
    C.setAxis(Color.White)


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