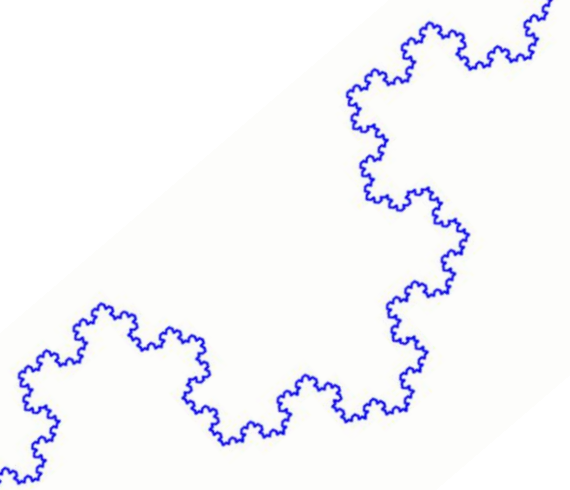
13. 5. 2019.

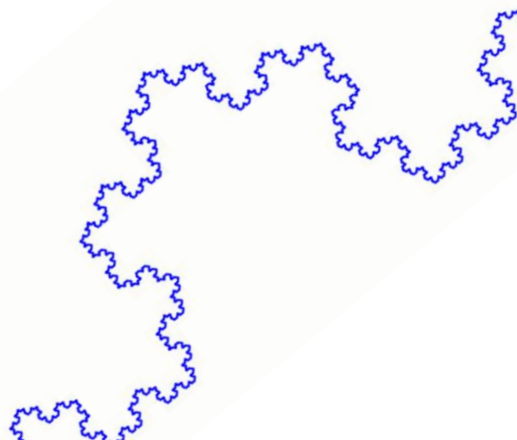


# Simulacija fraktala u Python-u

Predrag Mitić 116/2017

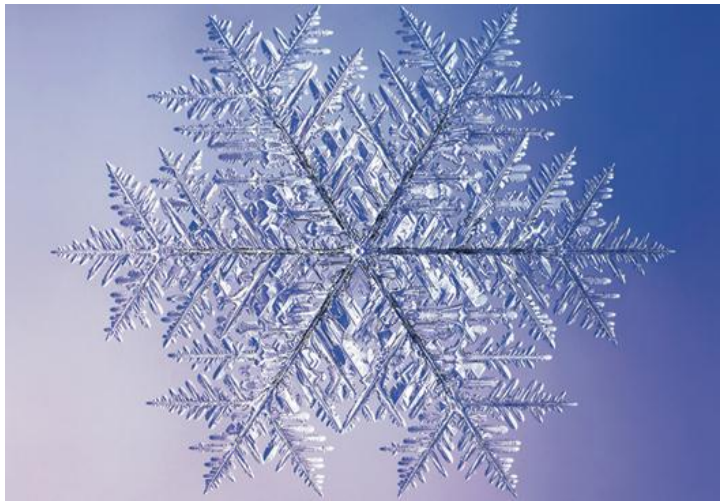
Đorđe Mutavdžić 96/2017

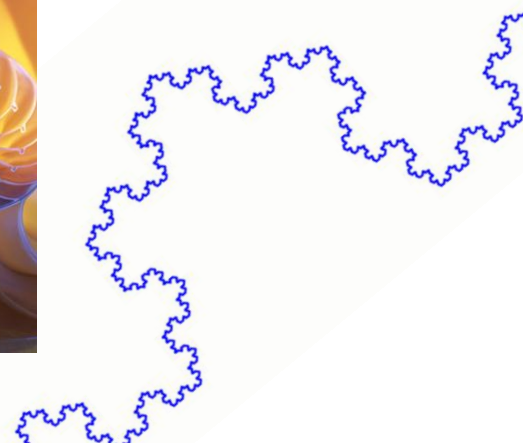
Ognjen Stamenković 64/2017



# Fraktali

- Geometrijski lik
- Samosličnost
- Fraktali u prirodi





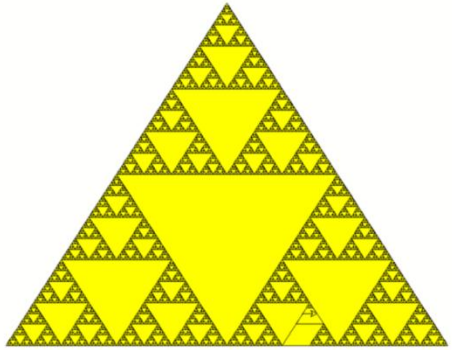
# Vrste fraktala

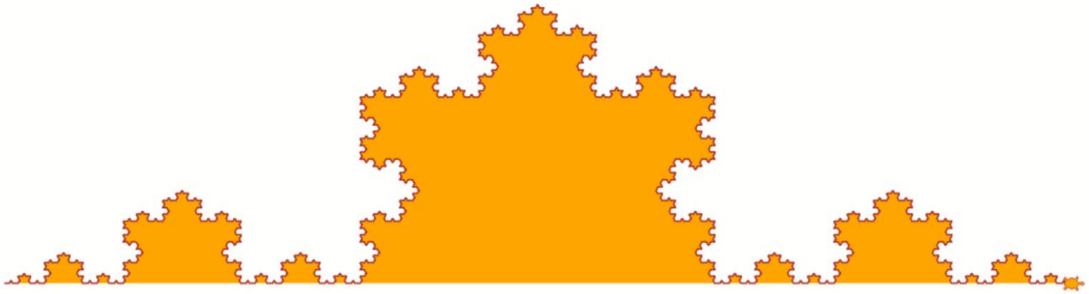
- Geometrijski
  - iterativna funkcije, potpuno samoslični
- Algebarski
  - rekurentne veze, skoro samoslični
- Stohastički
  - slučajni fraktali

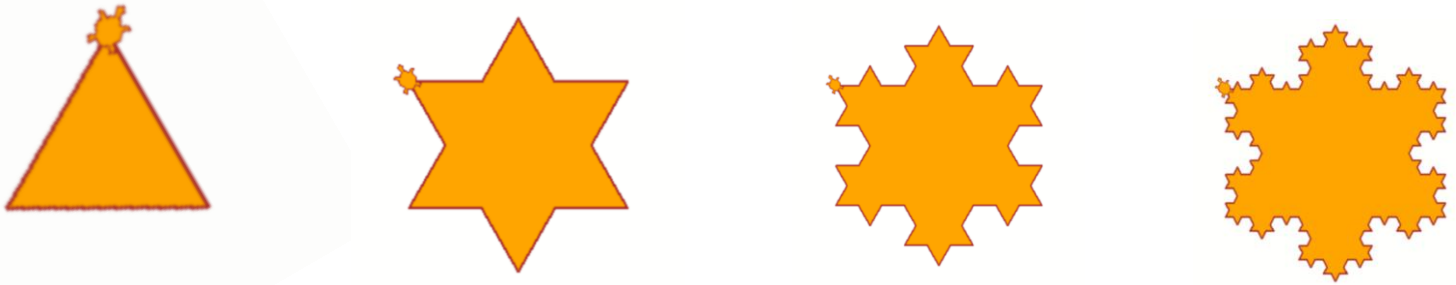
# Geometrijski fraktali

Poznatiji geometrijski fraktali

- Kohova pahulja
- Šerpinskijev trougao







Prve četiri iteracije Kohove krive

# Program koji simulira Kohovu krivu

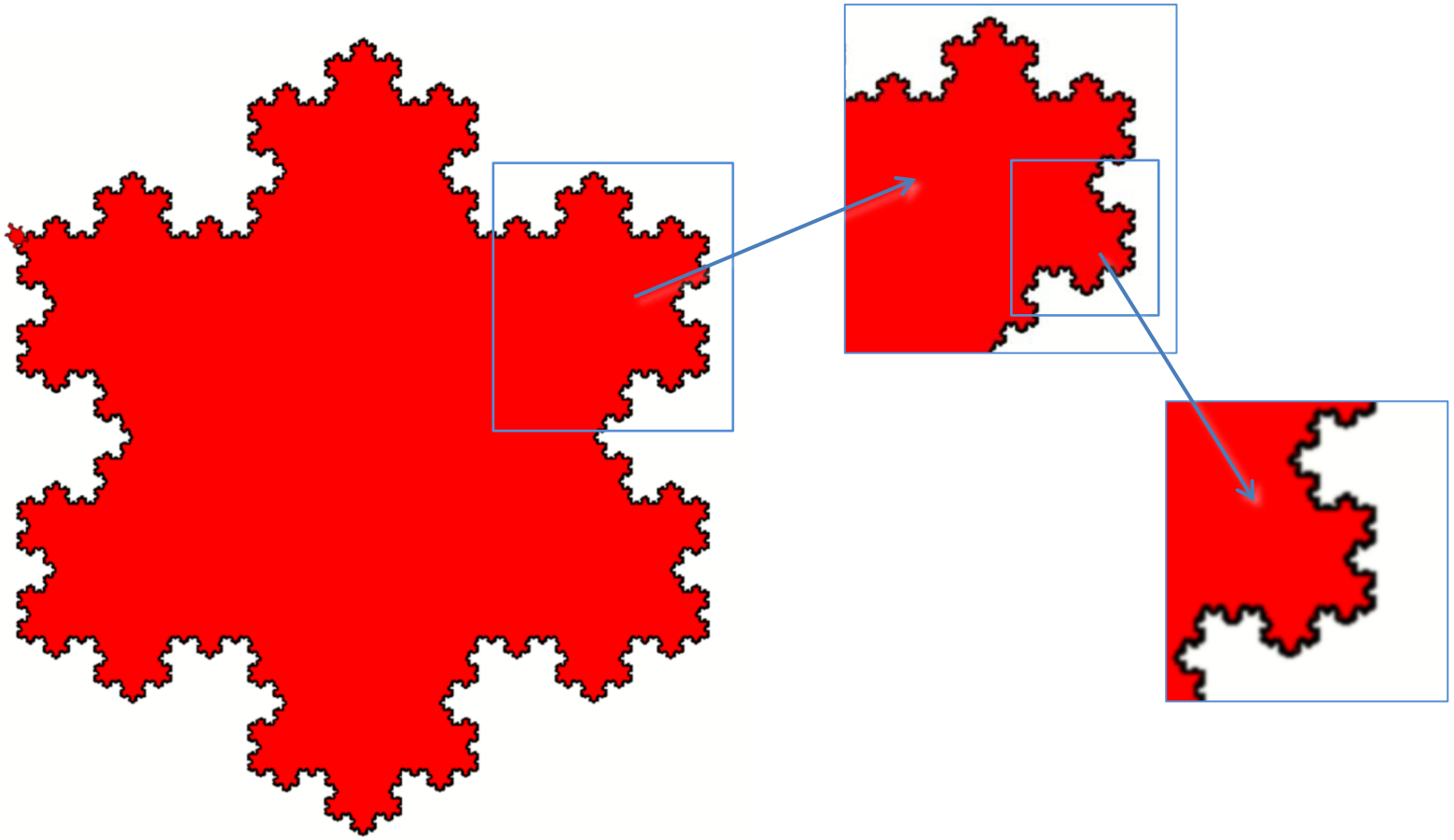
zvezda.py | trouglovi.py

```
1  import turtle
2
3  t = turtle.Turtle()
4  t.color("white")
5  t.pensize(3)
6  t.shape('turtle')
7  t.speed(0)
8  t.goto(-500,-200)
9
10 def fraktalZvezda(i, k):
11     if k == 0:
12         t.forward(i)
13     else:
14         fraktalZvezda(i, k-1)
15         t.left(60)
16         fraktalZvezda(i, k-1)
17         t.right(120)
18         fraktalZvezda(i, k-1)
19         t.left(60)
20         fraktalZvezda(i, k-1)
21
```

zvezda.py | trouglovi.py

```
22 def nacrtajFraktal(i,n):
23     fraktalZvezda(i,n)
24     t.right(120)
25     fraktalZvezda(i,n)
26     t.right(120)
27     fraktalZvezda(i,n)
28
29
30
31 korak = int(input("unesite korak: "))
32 stepen = int(input('Unesi stepen: '))
33
34 kraj = turtle.Screen()
35
36 t.color('blue', 'white')
37 t.begin_fill()
38 nacrtajFraktal(korak,stepen)
39 t.end_fill()
40
41 kraj.exitonclick()
42
```

# Rezultati simulacija



# Simulacija Šerpinskijevog Trougla

zvezda.py | trouglovi.py

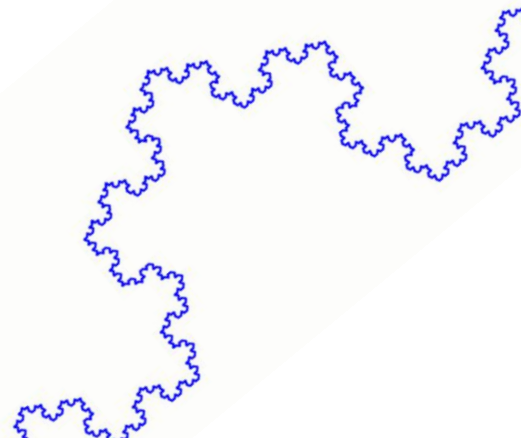
```
1  import turtle
2
3  def nacrtajTrougao(tacka, t):
4      t.up()
5      t.begin_fill()
6      t.goto(tacka[0][0], tacka[0][1])
7      t.down()
8      t.goto(tacka[1][0], tacka[1][1])
9      t.goto(tacka[2][0], tacka[2][1])
10     t.goto(tacka[0][0], tacka[0][1])
11     t.end_fill()
12
13 def sredina(p1,p2):
14     return ((p1[0] + p2[0]) / 2, (p1[1] + p2[1]) / 2)
15
```

zvezda.py | trouglovi.py

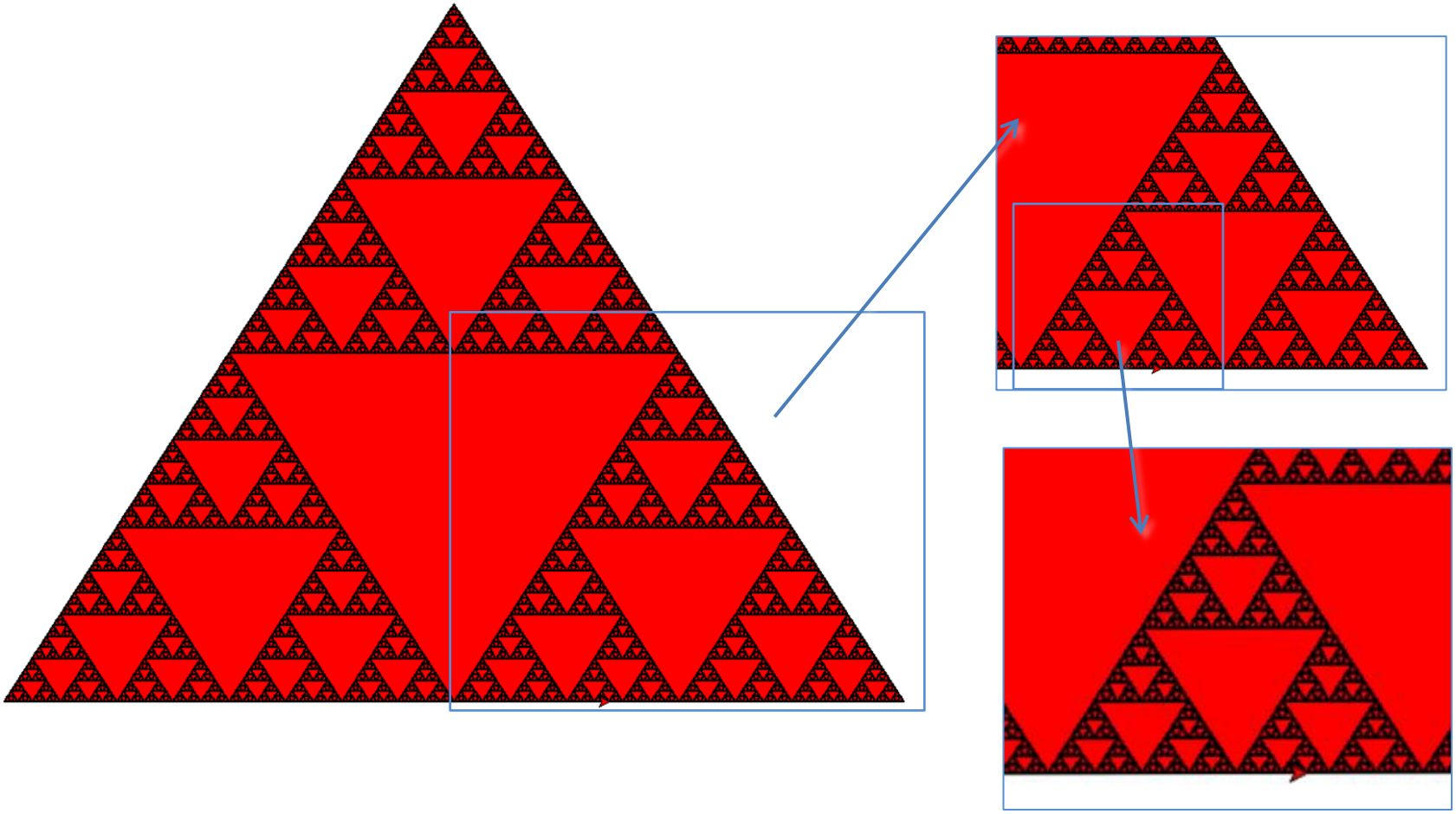
```
16 def fraktal(tacka, stepen, t):
17     nacrtajTrougao(tacka, t)
18     if stepen > 0:
19         fraktal([tacka[0],
20                  sredina(tacka[0], tacka[1]),
21                  sredina(tacka[0], tacka[2])],
22                 stepen - 1, t)
23         fraktal([tacka[1],
24                  sredina(tacka[0], tacka[1]),
25                  sredina(tacka[1], tacka[2])],
26                 stepen - 1, t)
27         fraktal([tacka[2],
28                  sredina(tacka[2], tacka[1]),
29                  sredina(tacka[0], tacka[2])],
30                 stepen - 1, t)
31
```

zvezda.py | trouglovi.py

```
32 def main():
33     t = turtle.Turtle()
34     t.color('black', 'yellow')
35     t.shape()
36     t.speed(10)
37     kraj = turtle.Screen()
38     k = input("unesi velicinu (500): ")
39     k = int(k)
40     tacke = [[-k*0.86,-k/1.5],[0,k/1.5],[k*0.86 ,-k/1.5
41                                          ]]
42     stepen = input("unesi stepen: ")
43     stepen = int(stepen)
44
45     fraktal(tacke,stepen,t)
46     kraj.exitonclick()
47
48 main()
49
```

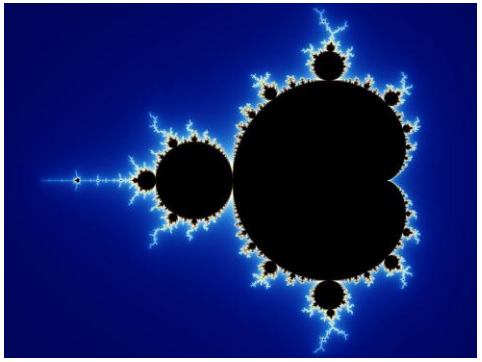


# Rezultati simulacija

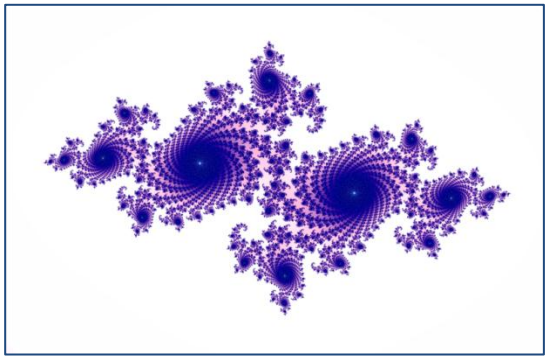


# Algebarski fraktali

Mandelbrotov skup



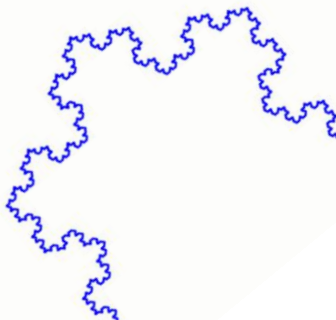
Džulijin skup



$$Z_{n+1} = Z_n^2 + C$$

$Z_0 = 0$
C – iz kompleksne ravni

$Z_0$ - iz kompleksne ravni
C – kompleksna konstanta

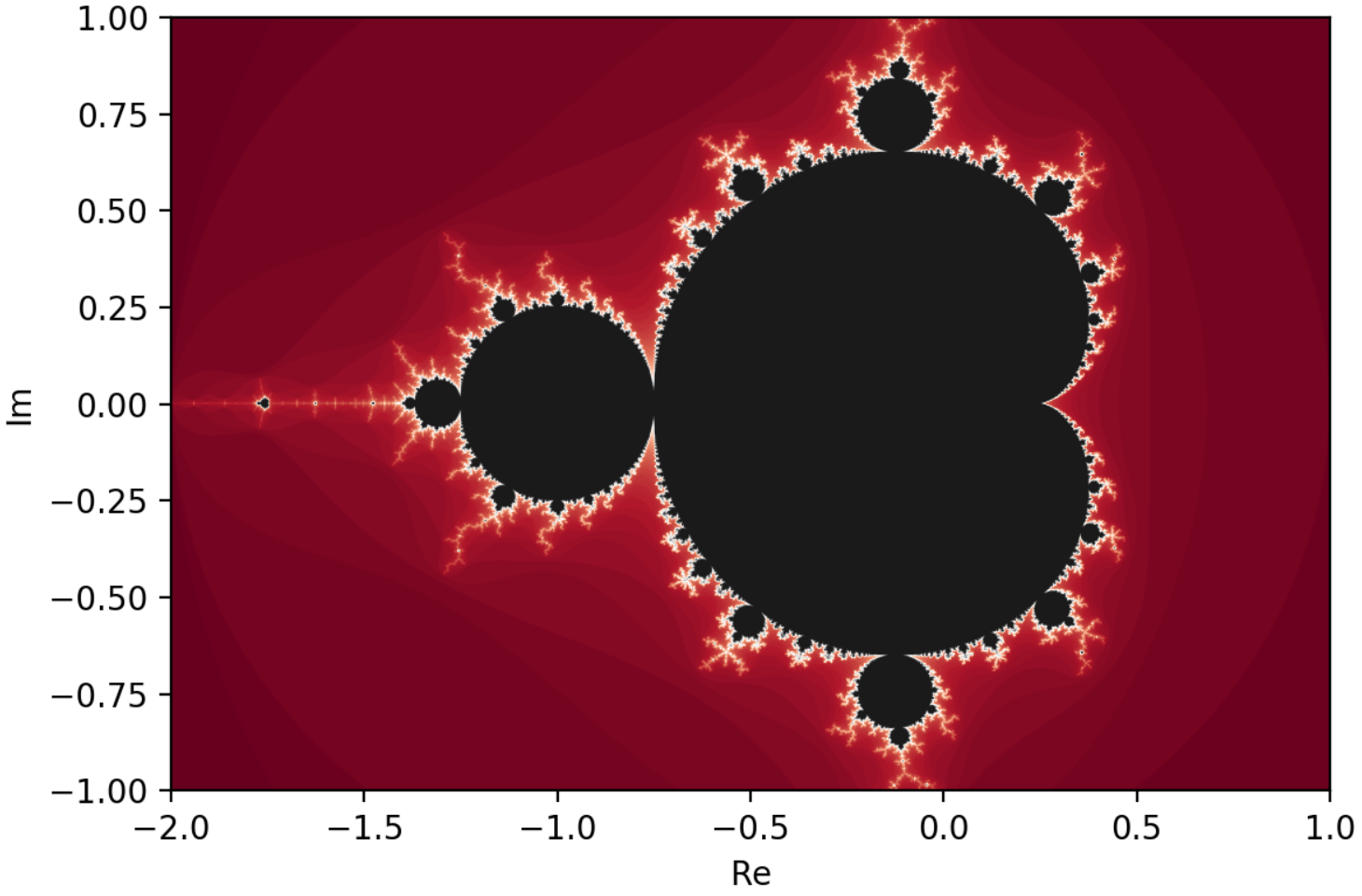


# Simulacija Mandelbrotovog skupa

MandelbrotSet.py

```
import numpy
import matplotlib.pyplot as plt

def mandelbrot(Re, Im, max_iter):
    c = complex(Re, Im)
    z = 0.0

    for i in range(max_iter):
        z = z*z + c
        if(z.real * z.real + z.imag * z.imag) >= 4:
            return i

    return max_iter

br_vrsta = 2000
br_kolona = 2000

rezultat = numpy.zeros([br_vrsta, br_kolona])

for index_vr, Re in enumerate(numpy.linspace(-2, 1, num=br_vrsta)):
    for index_kol, Im in enumerate(numpy.linspace(-1, 1, num=br_kolona)):
        rezultat[index_vr, index_kol] = mandelbrot(Re, Im, 100)

plt.figure(dpi=200)
plt.imshow(rezultat.T, cmap='RdGy', interpolation='bilinear', extent=[-2, 1, -1, 1])
plt.xlabel('Re')
plt.ylabel('Im')
plt.savefig('mandelb_rdgy.png')
```

# Rezultat simulacije

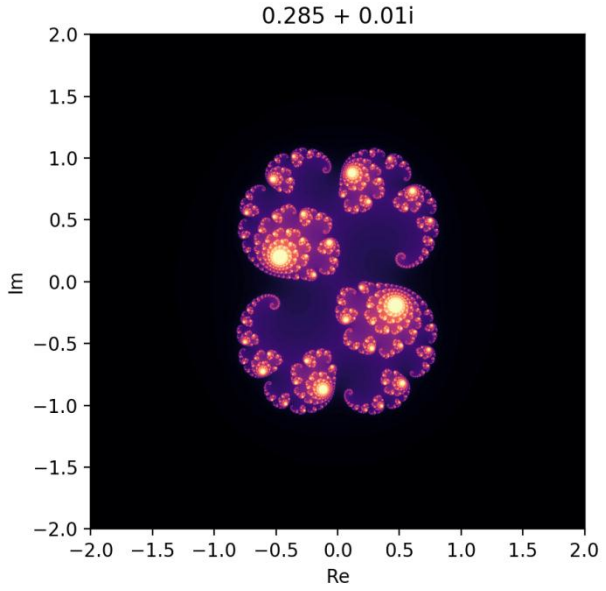


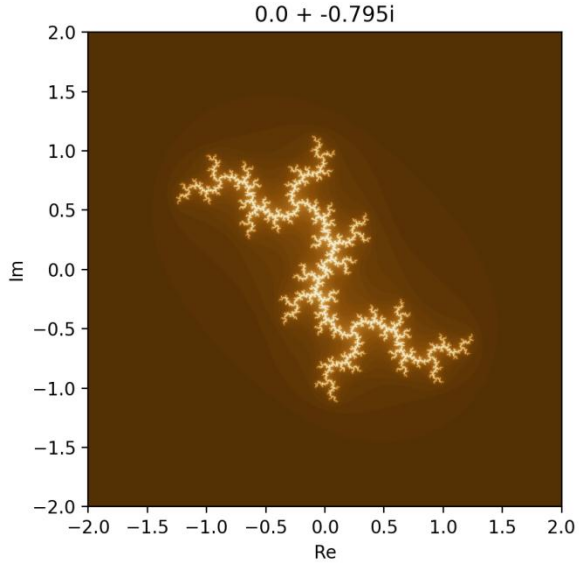
# Simulacija Džulijinog skupa

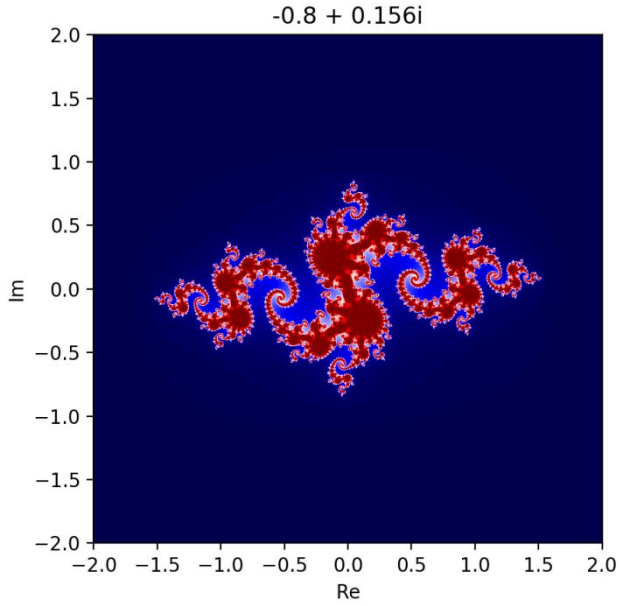
JuliaSet.py

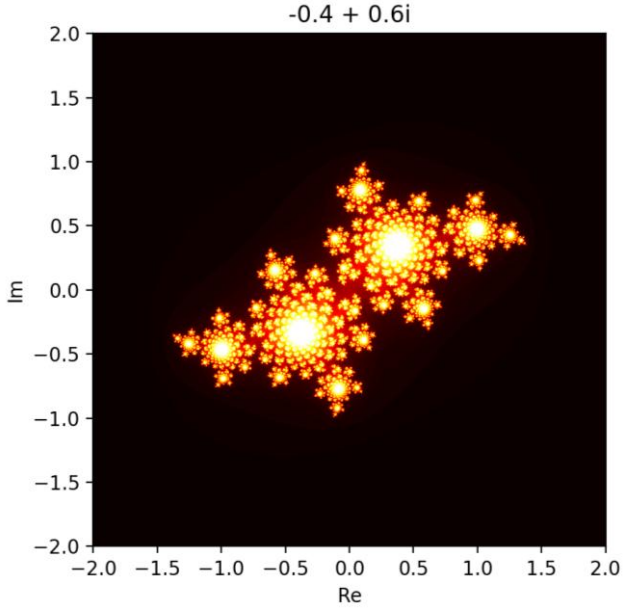
```
import numpy
import matplotlib.pyplot as plt

def julia(Re, Im, max_iter, ReC, ImC):
    z = complex(Re, Im)
    c = complex(ReC, ImC)

    for i in range(max_iter):
        z = z*z + c
        if(z.real * z.real + z.imag * z.imag) >= 4:
            return i

    return max_iter

br_vrsta = 2000
br_kolona = 2000

ReC = 0.0
ImC = -0.795

rezultat = numpy.zeros([br_vrsta, br_kolona])

for index_vr, Re in enumerate(numpy.linspace(-2, 2, num=br_vrsta)):
    for index_kol, Im in enumerate(numpy.linspace(-2, 2, num=br_kolona)):
        rezultat[index_vr, index_kol] = julia(Re, Im, 100, ReC, ImC)

plt.figure(dpi=200)
plt.imshow(rezultat.T, cmap='BrBG', interpolation='bilinear', extent=[-2, 2, -2, 2])
plt.title(str(ReC) + ' + ' + str(ImC) +'i')
plt.xlabel('Re')
plt.ylabel('Im')
plt.savefig('julia_new3.png')
```

# Rezultati simulacije

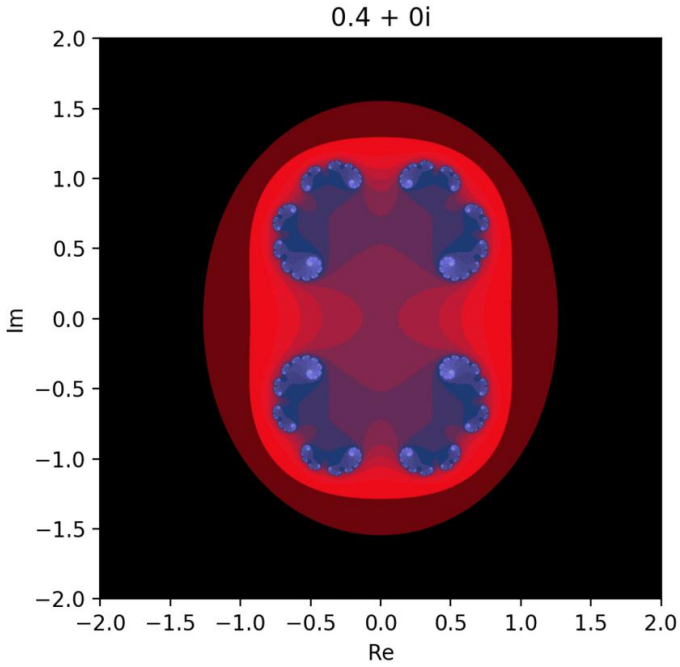


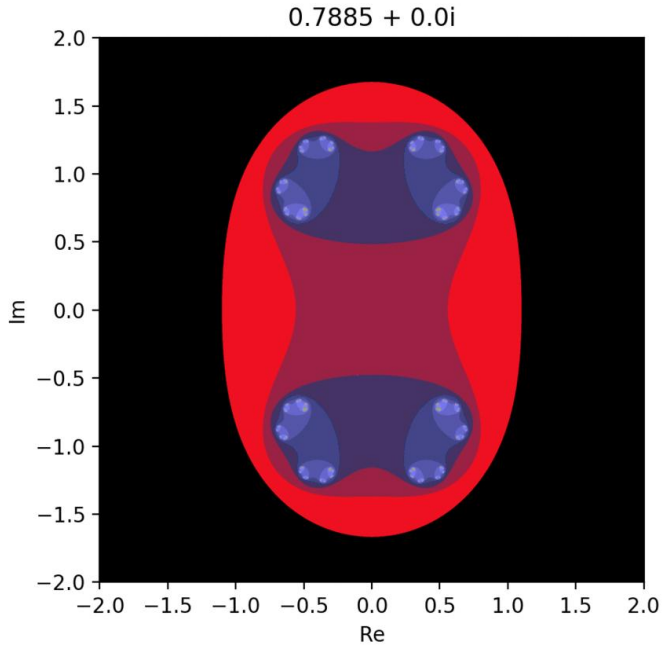



# Animacije Džulijinog skupa




Hvala na
pažnji