2. praktiskais darbs

1. uzdevums

Sastādīt programmu, kas aprēķina **N!** divos veidos – izmantojot ciklu un rekursiju.

Kods:

```
# Programmas nosaukums: Faktorials

# 1. uzdevums (1MPR02_Vladislavs_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas aprēķina N! divos veidos - izmantojot ciklu un rekursiju.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0



def is_natural_or_zero(n):
    # Pārbauda vai simbolu virkne ir naturāls skaitlis vai nulle, vai nav.
    # Ja ir naturāls skaitlis vai nulle, tad True. Ja nav tad False.
    # n - simbolu virkne (str), kuru pārbauda.
    if str(n).isdigit() and float(n) == int(n) and int(n) >= 0:
        return True
    else:
        return False



def factorial_cikls(n):
    # Aprēķina n faktoriāla vērtību izmantojot ciklu.
    # ja ir naturāls skaitlis vai nulle, tad return n!
    # ja nav naturāls skaitlis vai nulle, tad return False.
    # n - simbolu virkne (str).
    if is_natural_or_zero(n):
```

```
        n = int(n)
        a = 1
        while n >= 1:
            a = a * n
            n = n - 1
        return a
    else:
        return False


def factorial_rekursija(n):
    # Aprēķina n faktoriāla vērtību izmantojot rekursiju.
    # ja ir naturāls skaitlis vai nulle tad return n!
    # ja nav naturāls skaitlis vai nulle tad return False.
    # n - simbolu virkne (str).

    if is_natural_or_zero(n): # pārbauda vai ir naturāls skaitlis vai nulle, ja nav tad return False
        n = int(n)
        if n >= 0:
            if n == 0:
                return 1
            return factorial_rekursija(n - 1) * n
        else:
            return False
    else:
        return False


# ------------------------------------------------------
# Galvenā programmas daļa
# ------------------------------------------------------
```

```
n = input("Ievadiet naturālu skaitli vai nulli!\nN ==> ")
while factorial_cikls(n) == False or factorial_rekursija(n) == False:
    n = input("Kļūda! Tas nav naturāls skaitlis vai nulle.\nIevadiet naturālu skaitli vai nulli!\nN ==> ")
print(n + "! = " + str(factorial_cikls(n)) + " (ar ciklu)")
print(n + "! = " + str(factorial_rekursija(n)) + " (ar rekursiju)")
```

Testa piemēri:

1)

```
Ievadiet naturālu skaitli vai nulli!
N ==> 0
0! = 1 (ar ciklu)
0! = 1 (ar rekursiju)
```

2)

```
Ievadiet naturālu skaitli vai nulli!
N ==> 1
1! = 1 (ar ciklu)
1! = 1 (ar rekursiju)
```

3)

```
Ievadiet naturālu skaitli vai nulli!
N ==> Nulle
Kļūda! Tas nav naturāls skaitlis vai nulle.
Ievadiet naturālu skaitli vai nulli!
N ==> Pieci
Kļūda! Tas nav naturāls skaitlis vai nulle.
Ievadiet naturālu skaitli vai nulli!
N ==> 5
5! = 120 (ar ciklu)
5! = 120 (ar rekursiju)
```

4)

```
Ievadiet naturālu skaitli vai nulli!
N ==> 12.5
Kļūda! Tas nav naturāls skaitlis vai nulle.
Ievadiet naturālu skaitli vai nulli!
N ==> -2
Kļūda! Tas nav naturāls skaitlis vai nulle.
Ievadiet naturālu skaitli vai nulli!
N ==> 12414.-44214faf
Kļūda! Tas nav naturāls skaitlis vai nulle.
Ievadiet naturālu skaitli vai nulli!
N ==> labi
Kļūda! Tas nav naturāls skaitlis vai nulle.
Ievadiet naturālu skaitli vai nulli!
N ==> 55
55! = 12696403353658275925965100847566516959580321051449436762275840000000000000 (ar ciklu)
55! = 12696403353658275925965100847566516959580321051449436762275840000000000000 (ar rekursiju)
```

5)

```
Ievadiet naturālu skaitli vai nulli!
N ==> 100
100! = 933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000 (ar ciklu)
100! = 933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000 (ar rekursiju)
```

6)

```
Ievadiet naturālu skaitli vai nulli!
N ==> N
Kļūda! Tas nav naturāls skaitlis vai nulle.
Ievadiet naturālu skaitli vai nulli!
N ==> 12.5
Kļūda! Tas nav naturāls skaitlis vai nulle.
Ievadiet naturālu skaitli vai nulli!
N ==> 12.0
Kļūda! Tas nav naturāls skaitlis vai nulle.
Ievadiet naturālu skaitli vai nulli!
N ==> .2
Kļūda! Tas nav naturāls skaitlis vai nulle.
Ievadiet naturālu skaitli vai nulli!
N ==> 2.3
Kļūda! Tas nav naturāls skaitlis vai nulle.
Ievadiet naturālu skaitli vai nulli!
N ==> 12.0000
Kļūda! Tas nav naturāls skaitlis vai nulle.
Ievadiet naturālu skaitli vai nulli!
N ==> 12
12! = 479001600 (ar ciklu)
12! = 479001600 (ar rekursiju)
```

2. uzdevums

Sastādīt programmu, kas aprēķina $\boldsymbol{C_N^M}$, ja zināms, ka

$$C_n^m = C_{n-1}^{m-1} + C_{n-1}^m \quad \text{un} \quad C_n^0 = C_n^n = 1$$

Kods:

```
# Programmas nosaukums: Kombinācijas

# 2. uzdevums (1MPR02_Vladislavs_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas aprēķina C(n,m) izmantojot ciklu un rekursiju.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0



def is_natural_or_zero(n):
    # Pārbauda vai simbolu virkne ir naturāls skaitlis vai nulle, vai nav.
    # Ja ir naturāls skaitlis vai nulle, tad True. Ja nav tad False.
    # n - simbolu virkne (str), kuru pārbauda.
    if str(n).isdigit() and float(n) == int(n) and int(n) >= 0:
        return True
    else:
        return False



def kombinacijas_cikls(n, m):
    # Aprēķina kombinācijas skaitu C(n,m) izmantojot ciklu. n >= m
    # n - naturāls skaitlis vai nulle (no cik) n ir "apakšā"
    # m - naturāls skaitlis vai nulle (pa cik) m ir "augšā"


        if m > n:
            return False
```

```
    elif m == 0 or m == n:
        return 1

    fn = 1
    for i in range(2, n + 1):
        fn = fn * i

    fm = 1
    for i in range(2, m + 1):
        fm = fm * i

    fnm = 1
    for i in range(2, n - m + 1):
        fnm = fnm * i

    return fn / fm / fnm


def kombinacijas_rekursija(n, m):
    # Aprēķina kombinācijas skaitu C(n,m) izmantojot rekursiju. n >= m
    # n - naturāls skaitlis vai nulle (no cik) n ir "apakšā"
    # m - naturāls skaitlis vai nulle (pa cik) m ir "augšā"
    if m > n:
        return False

    if m == 0 or m == n:
        return 1
    return kombinacijas_rekursija(n - 1, m - 1) + kombinacijas_rekursija(n - 1, m)
```

```
# ------------------------------------------------------------

# Galvenā programmas daļa

# ------------------------------------------------------------



n = input("Ievadiet naturalo skaitli vai nulli!\nn ==> ")


if not is_natural_or_zero(n):

    print("Kļūda! Tika ievadīti nekorekti dati!")

    quit()


m = input("Ievadiet naturalo skaitli vai nulli!\nm ==> ")


if not is_natural_or_zero(m):

    print("Kļūda! Tika ievadīti nekorekti dati!")

    quit()


n = int(n)

m = int(m)


if kombinacijas_cikls(n, m) == False or kombinacijas_rekursija(n, m) == False:

    print("Kļūda! m < n")

else:

    print("C(" + str(n) + ", " + str(m) + ") = " + str(int(kombinacijas_cikls(n, m))) + " (ar ciklu)")

    print("C(" + str(n) + ", " + str(m) + ") = " + str(kombinacijas_rekursija(n, m)) + " (ar
rekursiju)")
```

Testa piemēri:

1)

```
Ievadiet naturalo skaitli vai nulli!
n ==> 5
Ievadiet naturalo skaitli vai nulli!
m ==> 5
C(5, 5) = 1 (ar ciklu)
C(5, 5) = 1 (ar rekursiju)
```

2)

```
Ievadiet naturalo skaitli vai nulli!
n ==> 10
Ievadiet naturalo skaitli vai nulli!
m ==> 5
C(10, 5) = 252 (ar ciklu)
C(10, 5) = 252 (ar rekursiju)
```

3)

```
Ievadiet naturalo skaitli vai nulli!
n ==> 0
Ievadiet naturalo skaitli vai nulli!
m ==> 0
C(0, 0) = 1 (ar ciklu)
C(0, 0) = 1 (ar rekursiju)
```

4)

```
Ievadiet naturalo skaitli vai nulli!
n ==> 10
Ievadiet naturalo skaitli vai nulli!
m ==> 20
Kļūda! m < n
```

5)

```
Ievadiet naturalo skaitli vai nulli!
n ==> Pieci
Kļūda! Tika ievadīti nekorekti dati!
```

6)

```
Ievadiet naturalo skaitli vai nulli!
n ==> 5
Ievadiet naturalo skaitli vai nulli!
m ==> Desmit
Kļūda! Tika ievadīti nekorekti dati!
```

7)

```
Ievadiet naturalo skaitli vai nulli!
n ==> 12.4
Kļūda! Tika ievadīti nekorekti dati!
```

8)

```
Ievadiet naturalo skaitli vai nulli!
n ==> 12
Ievadiet naturalo skaitli vai nulli!
m ==> Trīs
Kļūda! Tika ievadīti nekorekti dati!
```

9)

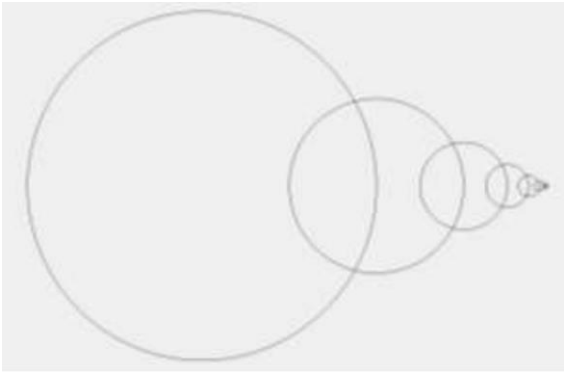
```
Ievadiet naturalo skaitli vai nulli!
n ==> 20
Ievadiet naturalo skaitli vai nulli!
m ==> 10
C(20, 10) = 184756 (ar ciklu)
C(20, 10) = 184756 (ar rekursiju)
```

10)

```
Ievadiet naturalo skaitli vai nulli!
n ==> 1000000000000
Ievadiet naturalo skaitli vai nulli!
m ==> 0
C(1000000000000, 0) = 1 (ar ciklu)
C(1000000000000, 0) = 1 (ar rekursiju)
```

3. uzdevums

Sastādīt programmu, kas izveido zemāk redzamo attēlu, izmantojot rekursiju.



Kods:

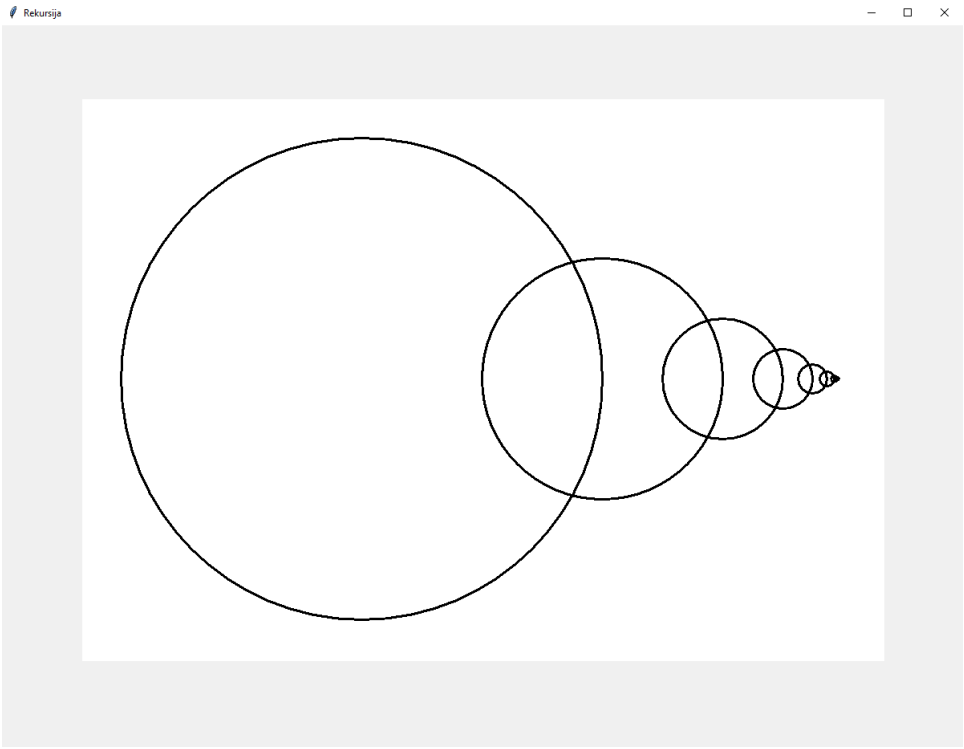
```
# Programmas nosaukums: Rekursija. Riņķa līnijas.

# 3. uzdevums (1MPR02_Vladislavs_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas izveido zemāk redzamo attēlu, izmantojot rekursiju.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0



import tkinter

from tkinter import ttk



def create_circle(x, y, r):

    # Uzzīme riņķa līniju

    # r - riņķa līnijas rādiuss

    # x - x koordināta riņķa līnijas centram

    # y - y koordināta riņķa līnijas centram

    kanva.create_oval(x - r, y - r, x + r, y + r, outline="black", width=3)



def draw_circles(x, y, r): # r - rādiuss
```

```
    # Uzzīme riņķa līnijas rekursīvi uz labo pusi
    # r - riņķa līnijas rādiuss
    # x - x koordināta riņķa līnijas centram
    # y - y koordināta riņķa līnijas centram
    if r >= 2:
        create_circle(x, y, r)
        draw_circles(x + r, y, r // 2)


# ------------------------------------------------------------
# Galvenā programmas daļa
# ------------------------------------------------------------


logs = tkinter.Tk()
logs.geometry("1200x900")
logs.title("Rekursija")

kanva = tkinter.Canvas(logs, bg="white", height=700, width=1000)
kanva.place(x=100, y=90)

draw_circles(350, 350, 300)

logs.mainloop()
```

Testa piemēri:

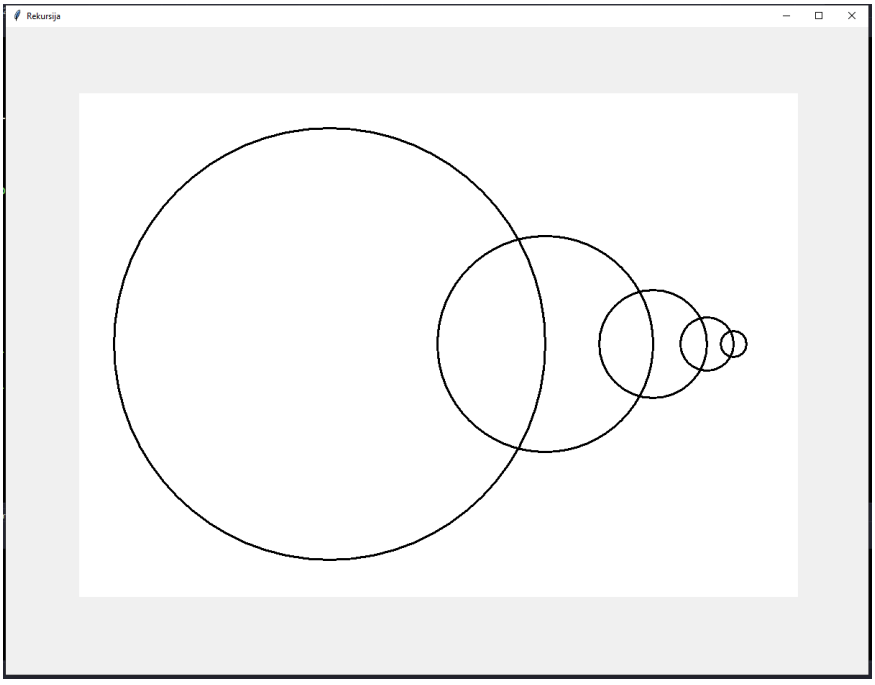
1) if **r >= 2**:

```
create_circle(x, y, r)
draw_circles(x + r, y, r // 2)
```



2) if **r >= 10**:

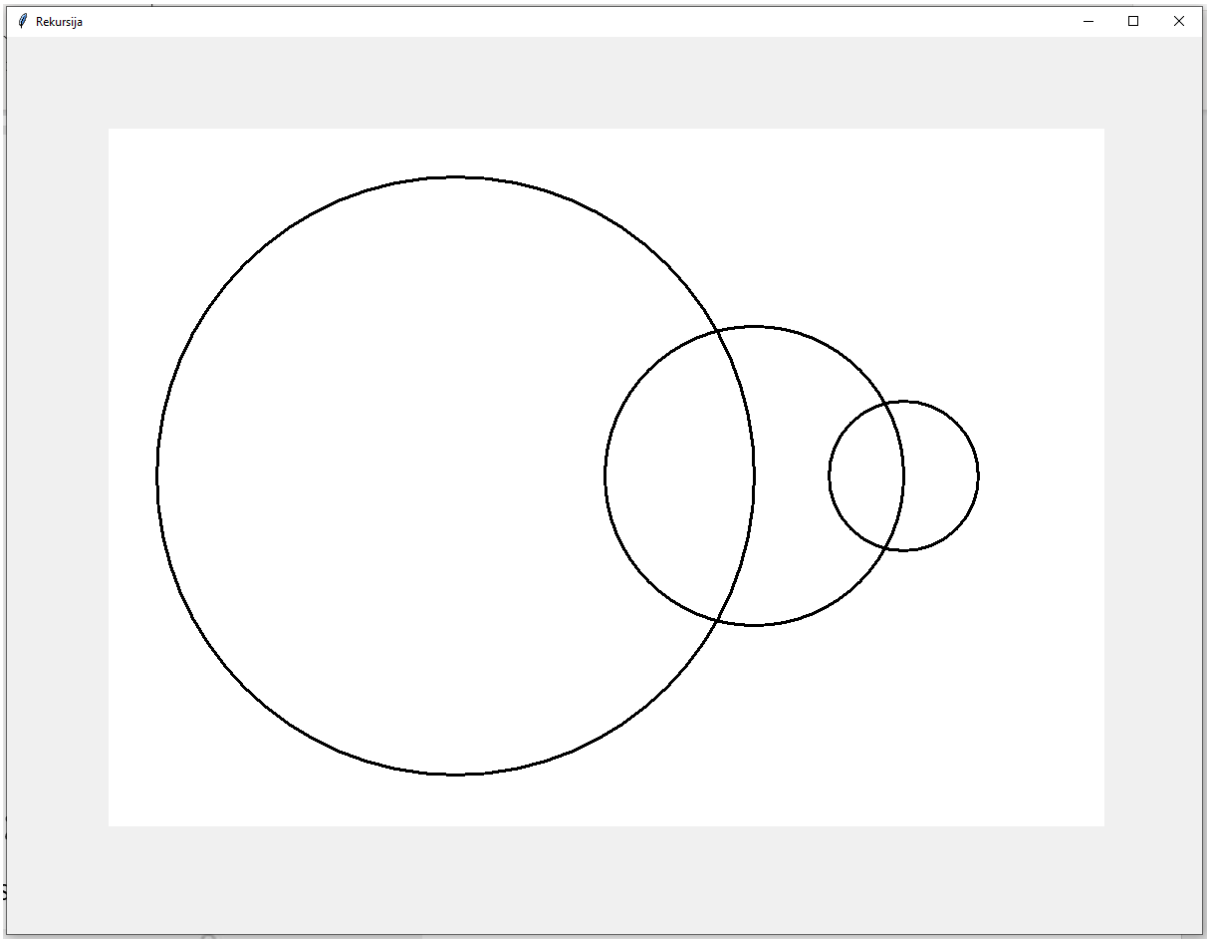
```
create_circle(x, y, r)

draw_circles(x + r, y, r // 2)
```



3) if **r >= 40**:

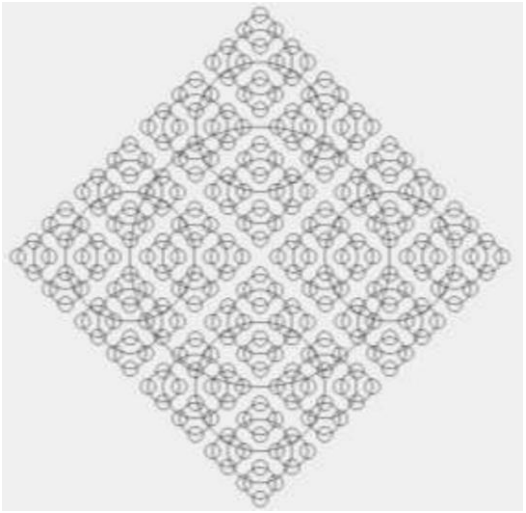
create_circle(x, y, r)

draw_circles(x + r, y, r // 2)



4. uzdevums

Sastādīt programmu, kas izveido zemāk redzamo attēlu, izmantojot rekursiju.



Kods:

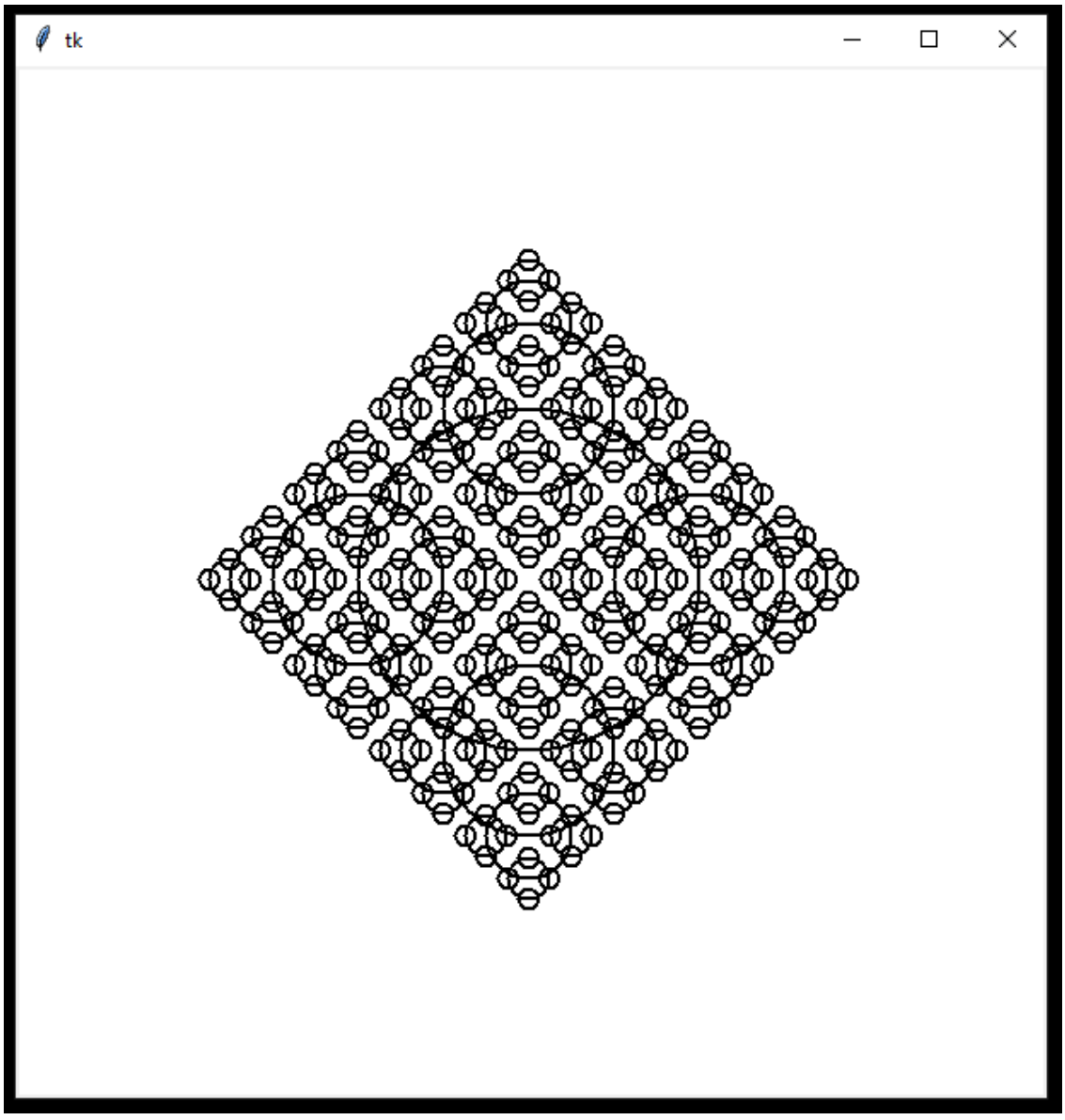
```
# Programmas nosaukums: Rekursija. Riņķa līnijas četri virzieni.

# 4. uzdevums (1MPR02_Vladislavs_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas izveido zemāk redzamo attēlu, izmantojot rekursiju.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0


import tkinter


logs = tkinter.Tk()

canva = tkinter.Canvas(logs, bg="white", height=600, width=600)

canva.pack()



def circles_four_directions(x, r, y):

    # Uzzīme riņķa līnijas četros virzienos

    # x - x koordināta riņķa līnijas centram

    # r - r riņķa līnijas rādiuss

    # y - y koordināta riņķa līnijas centram

    if r <= 3:

        return

    canva.create_oval(x - r, y - r, x + r, y + r, width=2)

    circles_four_directions(x + r, r // 2, y)

    circles_four_directions(x - r, r // 2, y)

    circles_four_directions(x, r // 2, y - r)

    circles_four_directions(x, r // 2, y + r)
```

```
# ------------------------------------------------------------

# Galvenā programmas daļa

# ------------------------------------------------------------


circles_four_directions(300, 100, 300)


logs.mainloop()
```

Testa piemēri:

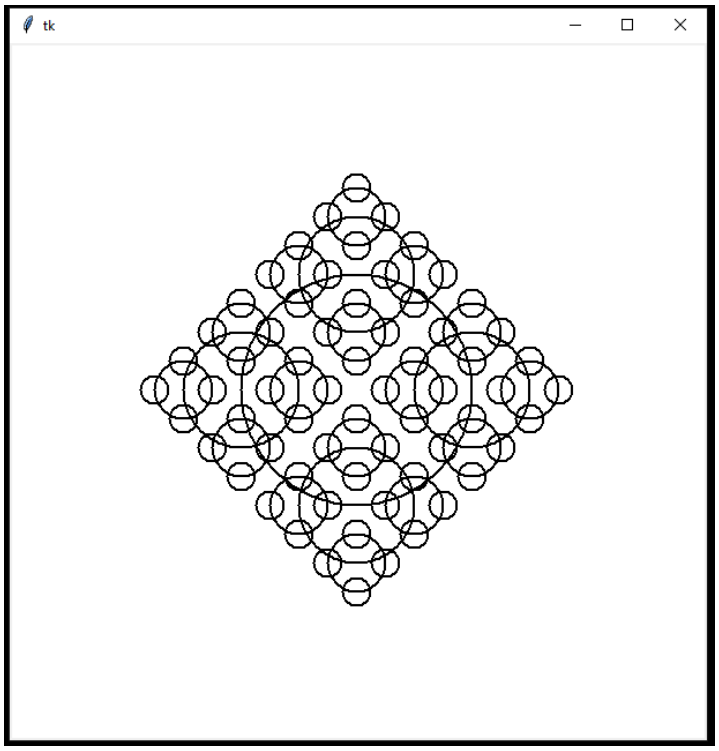
1) if **r <= 3**:

   return…



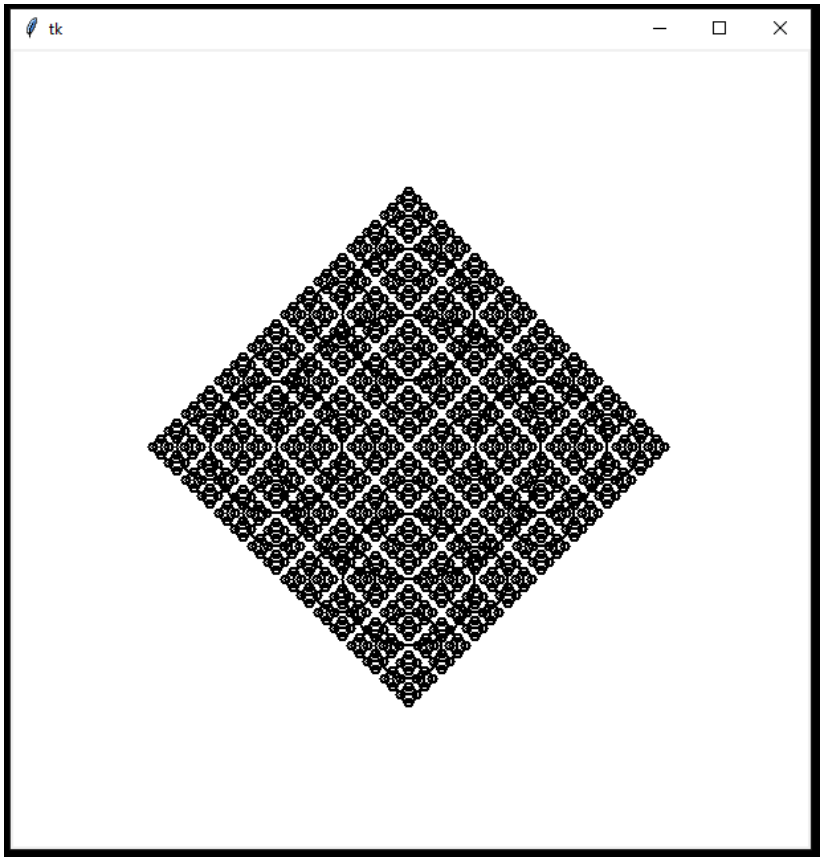
2) if **r <= 10**:

return…



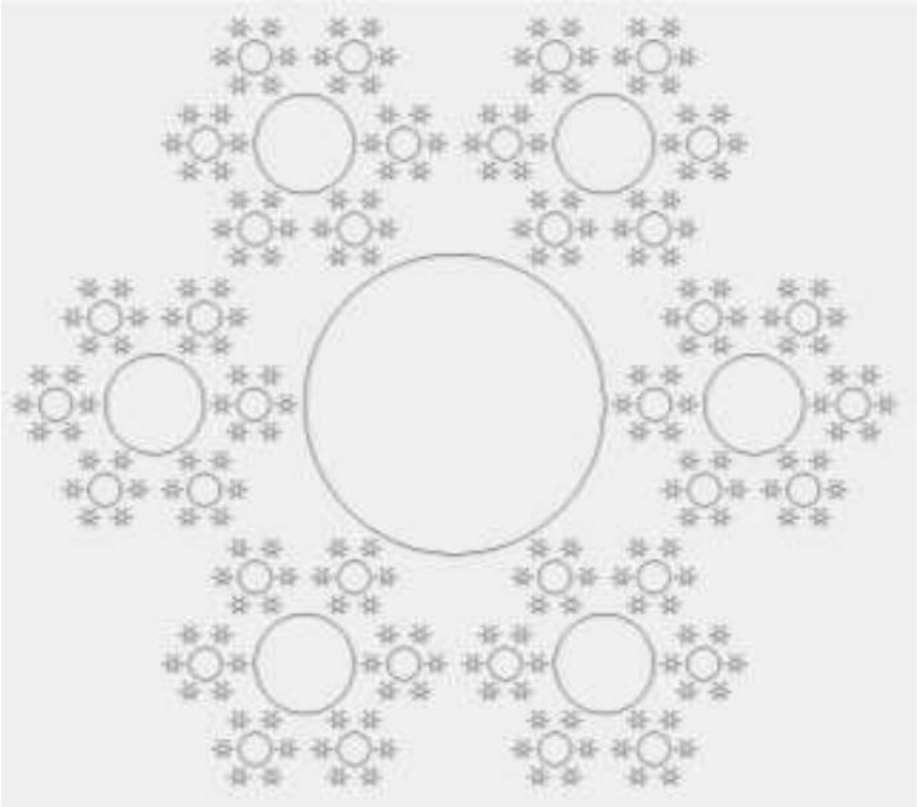
3) if **r <= 2**:

return…



5. uzdevums

Sastādīt programmu, kas izveido zemāk redzamo attēlu, izmantojot rekursiju.



Kods:

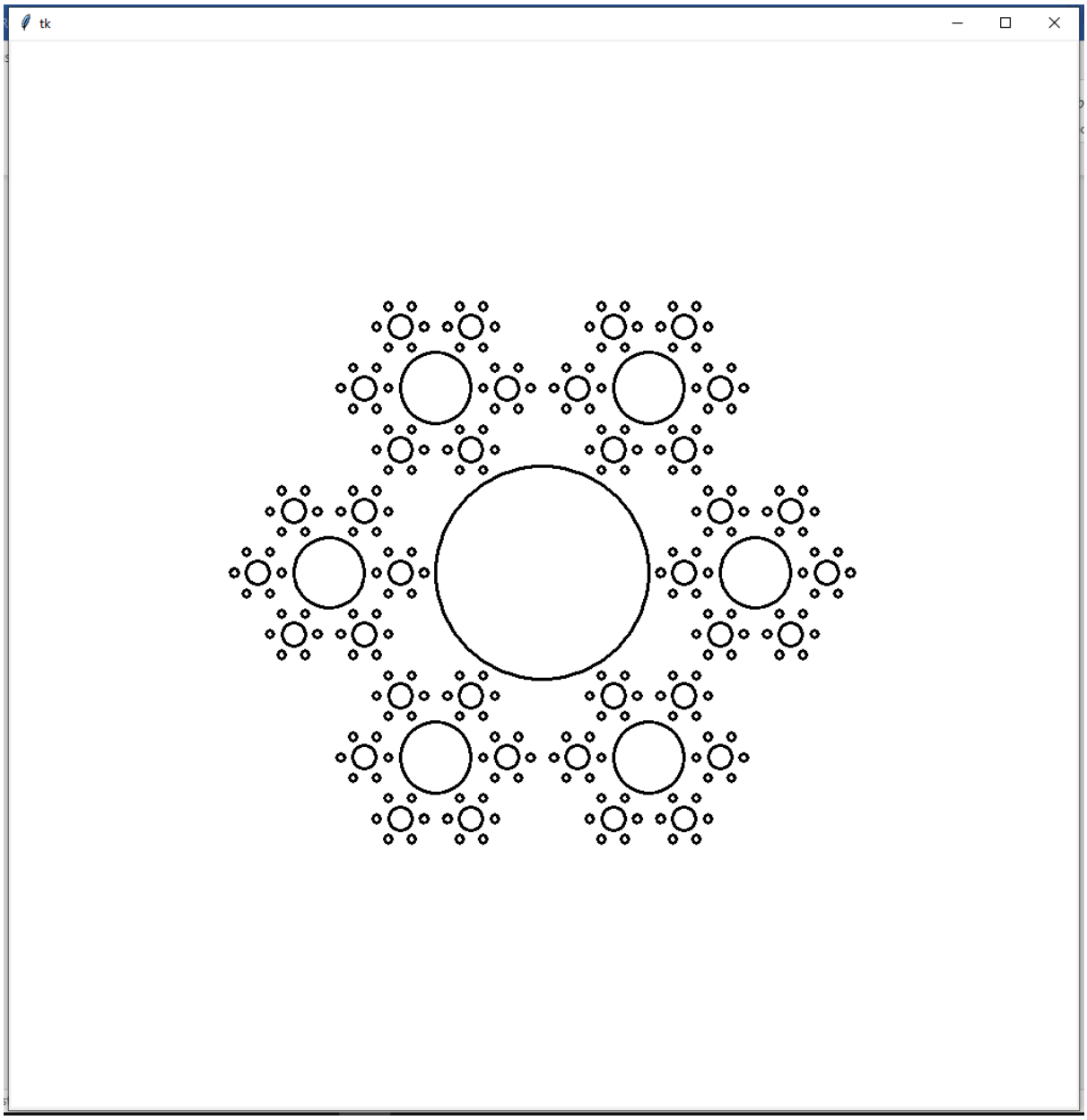
```
# Programmas nosaukums: Rekursija. Riņķa līnijas sešos virzienos.

# 5. uzdevums (1MPR02_Vladislavs_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas izveido zemāk redzamo attēlu, izmantojot rekursiju.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0



import tkinter

import math


logs = tkinter.Tk()

canva = tkinter.Canvas(logs, bg="white", height=1000, width=1000)

canva.pack()
```

```
def circles_six_directions(x, r, y):
    # Uzzīme riņķa līnijas sešos virzienos
    # x - x koordināta riņķa līnijas centram
    # r - r riņķa līnijas rādiuss
    # y - y koordināta riņķa līnijas centram
    if r <= 2:
        return
    canva.create_oval(x - r, y - r, x + r, y + r, width=3)

    circles_six_directions(x + 2 * r, r / 3, y)
    circles_six_directions(x - 2 * r, r / 3, y)

    circles_six_directions(x + r, r / 3, y - r * math.sqrt(3))
    circles_six_directions(x - r, r / 3, y - r * math.sqrt(3))

    circles_six_directions(x + r, r / 3, y + r * math.sqrt(3))
    circles_six_directions(x - r, r / 3, y + r * math.sqrt(3))



# ------------------------------------------------------
# Galvenā programmas daļa
# ------------------------------------------------------



circles_six_directions(500, 100, 500)


logs.mainloop()
```

Testa piemēri:

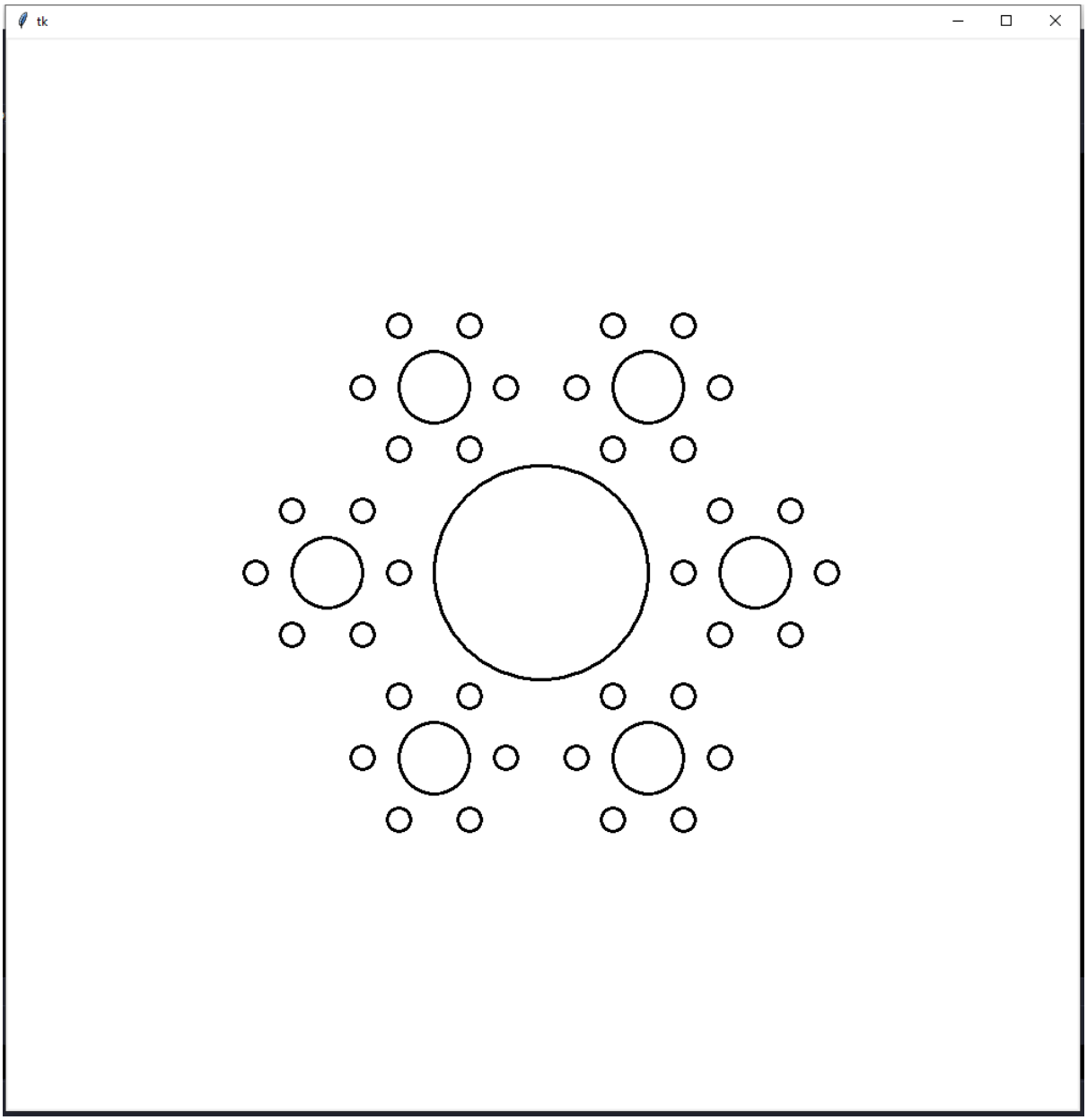
1) if **r <= 2**:

return…



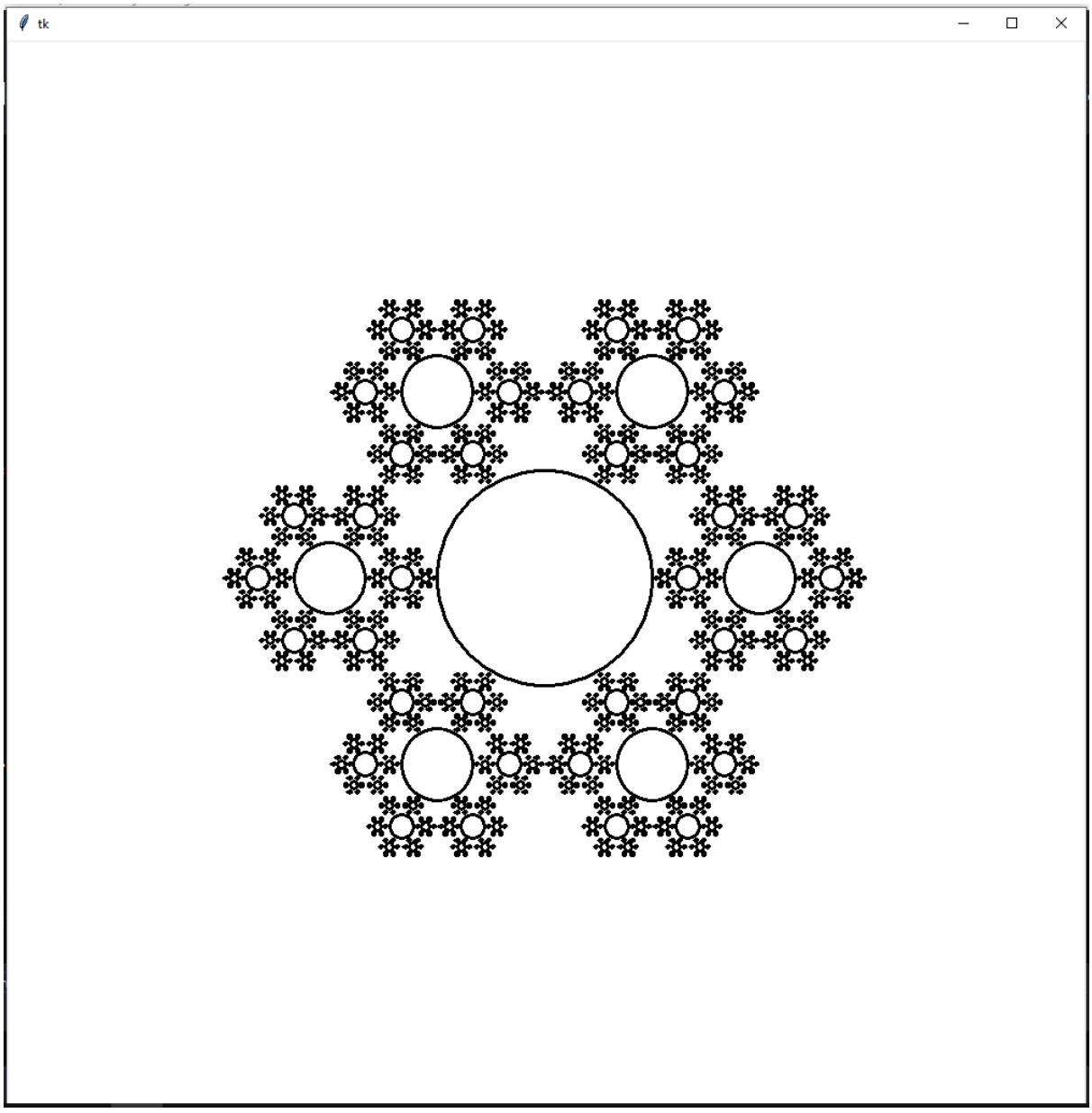
2) if **r <= 4**:

return



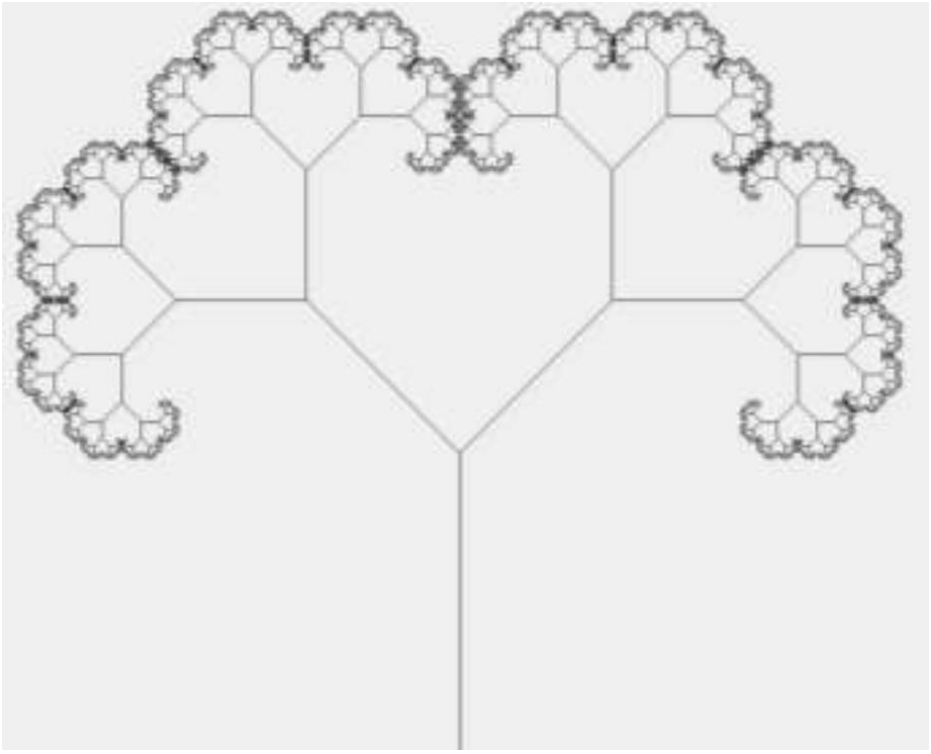
3) if **r <= 1**:

Return…



6. uzdevums

Sastādīt programmu, kas izveido zemāk redzamo attēlu, izmantojot rekursiju.



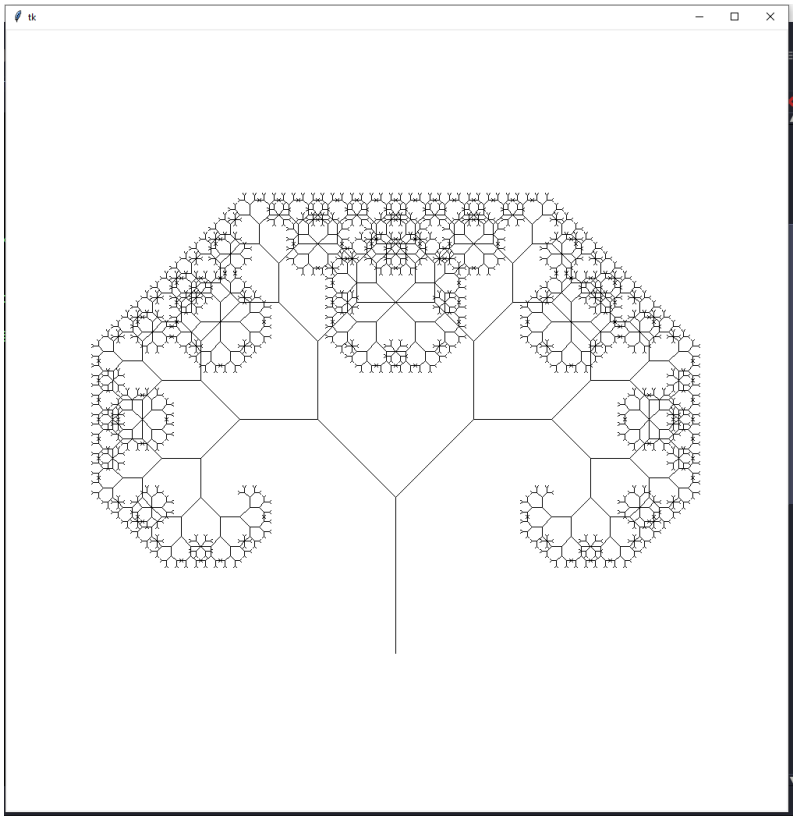
Kods:

```
# Programmas nosaukums: Rekursija. Pitagora koks.

# 6. uzdevums (1MPR02_Vladislavs_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas izveido zemāk redzamo attēlu, izmantojot rekursiju.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0



import tkinter

import math



logs = tkinter.Tk()

canvas = tkinter.Canvas(logs, bg="white", height=1000, width=1000)

canvas.pack()
```

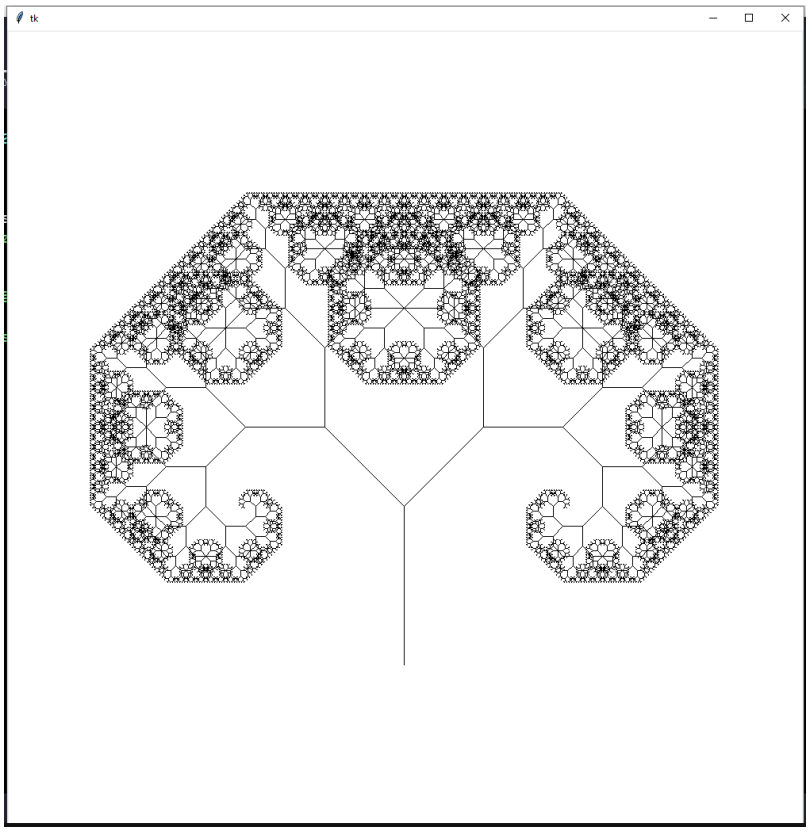
```
def zimet_pitagora_koku(x, y, garums, lenkis, rekursijas_skaits):
    # Uzzīme Pitagora koku noteiktā vietā, izmēra un tas tiek zimēts rekursīvi "rekursijas_skaits" reizes.
    # x - x koordināta pirmā nogriežņa sākumam
    # y - y koordināta pirmā nogriežņa beigām
    # garums - garums pirmām nogriežņim. Tāda veidā tiek regulēts Pitagora koka izmērs
    # lenkis - leņķis grādos pirmām nogriežņim ar "zēmi"
    # rekursijas_skaits - rekursijas skaits. Maksimālais skaits pēc kurā zimēšana beigsies
    if rekursijas_skaits > 0:
        # print(lenkis)
        x1 = x + garums * math.cos(math.radians(lenkis))
        y1 = y - garums * math.sin(math.radians(lenkis))
        canvas.create_line(x, y, x1, y1)

        zimet_pitagora_koku(x1, y1, garums * math.sqrt(2) / 2, lenkis - 45, rekursijas_skaits - 1)
        zimet_pitagora_koku(x1, y1, garums * math.sqrt(2) / 2, lenkis + 45, rekursijas_skaits - 1)


# ------------------------------------------------------
# Galvenā programmas daļa
# ------------------------------------------------------


zimet_pitagora_koku(500, 800, 200, 90, 12)

logs.mainloop()
```

Testa piemēri:

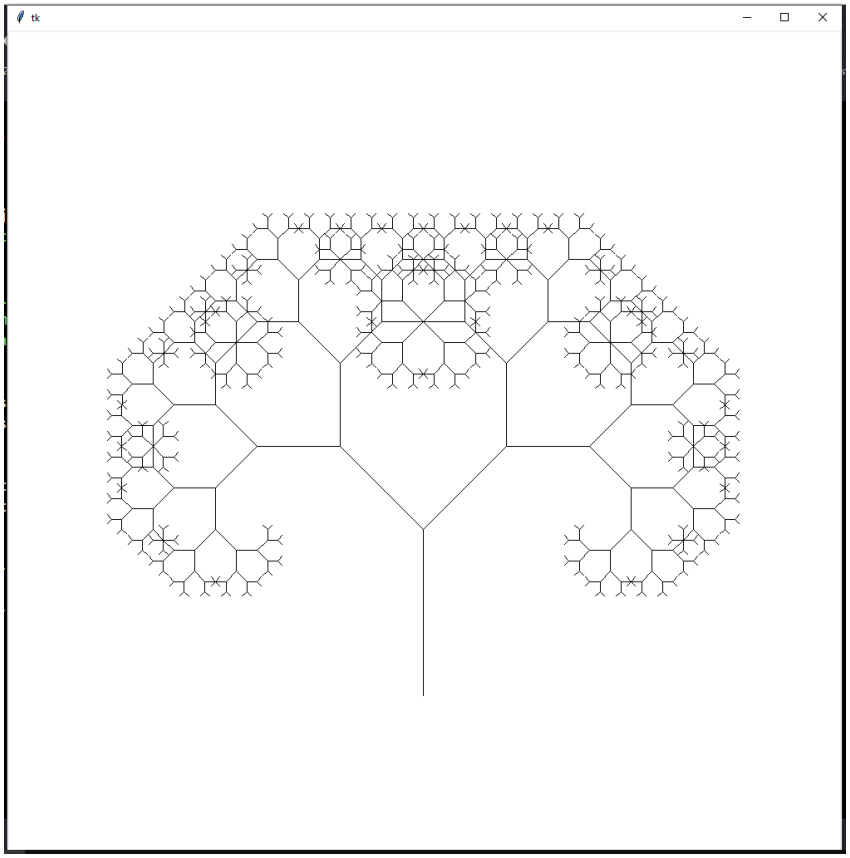
1) **rekursijas_skaits == 12**



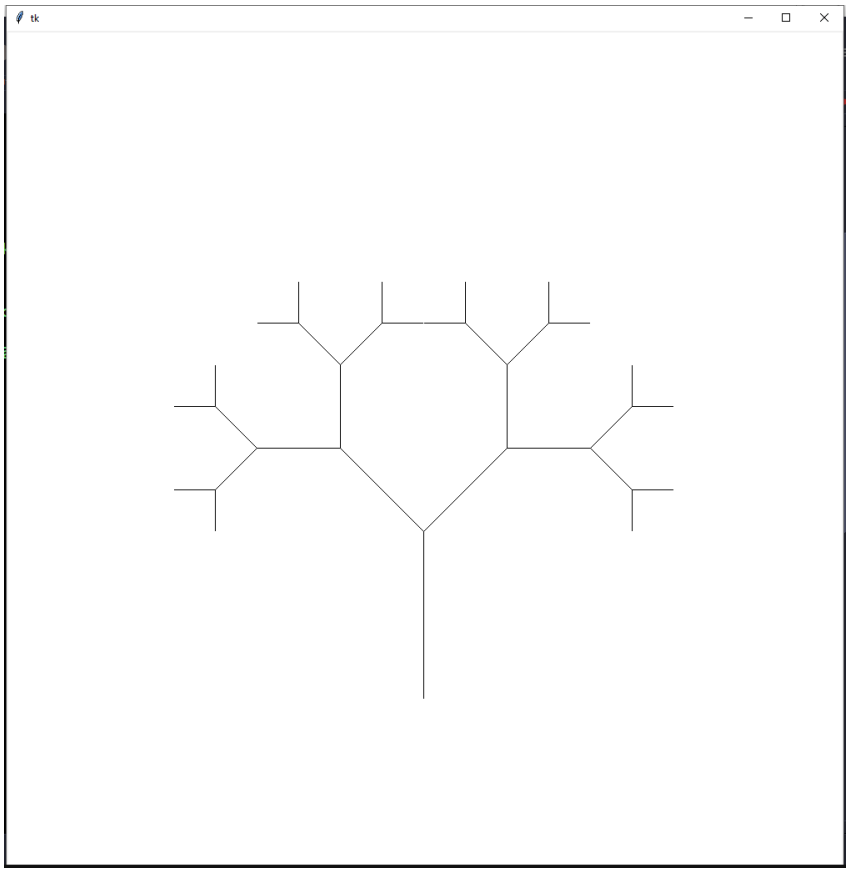
2) **rekursijas_skaits == 15**



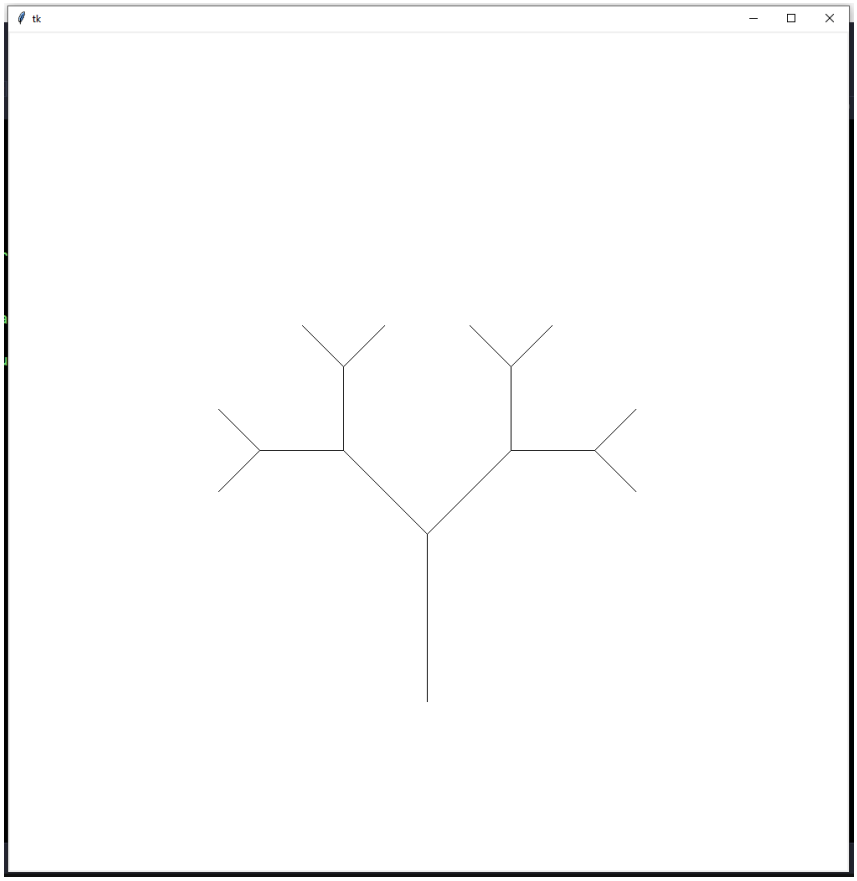
3) **rekursijas_skaits == 10**



4) **rekursijas_skaits == 5**

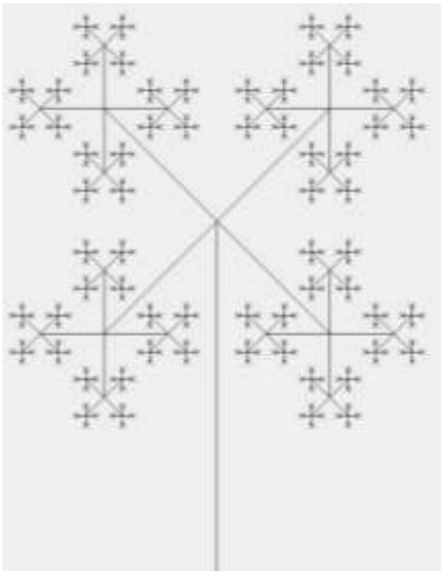


5) **rekursijas_skaits == 4**



PU1. uzdevums

Sastādīt programmu, kas izveido zemāk redzamo attēlu, izmantojot rekursiju.



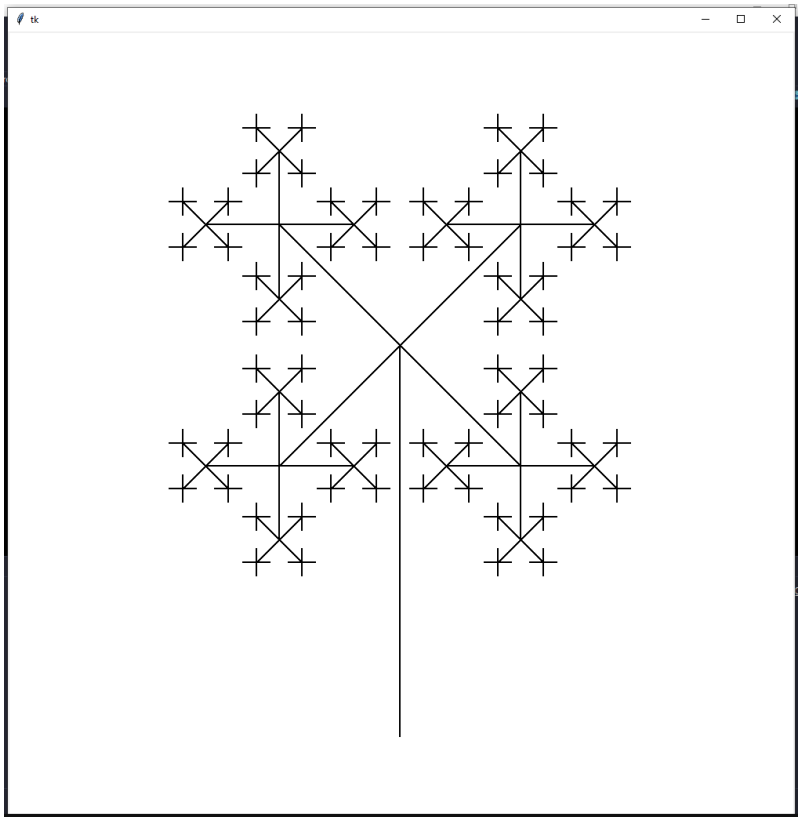
Kods:

```
# Programmas nosaukums: Rekursija. Krusti četros virzienos.

# Papilduzdevums 1. (1MPR02_Vladislavs_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas izveido zemāk redzamo attēlu, izmantojot rekursiju.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0



import tkinter

import math


logs = tkinter.Tk()

canvas = tkinter.Canvas(logs, bg="white", height=1000, width=1000)

canvas.pack()



def draw_line(x0, y0, x1, y1):

    # Uzzīme nogriežņi pēc diviem dotiem punktiem ar biezumu (width) 2

    # x0 - x koordināta sākumpunktam

    # y0 - y koordināta sākumpunktam

    # x1 - x koordināta galapunktam

    # y1 - y koordināta galapunktam

    canvas.create_line(x0, y0, x1, y1, width=2)



def draw_crosses(x0, y0, x1, y1, rekursijas_skaits):

    # Rekursīvi uzzīme krustiņus. Koeficienti tika iegūti aptuveni.

    # x0 - x koordināta sākumpunktam (nogriežņim)

    # y0 - y koordināta sākumpunktam (nogriežņim)

    # x1 - x koordināta galapunktam (nogriežņim)
```

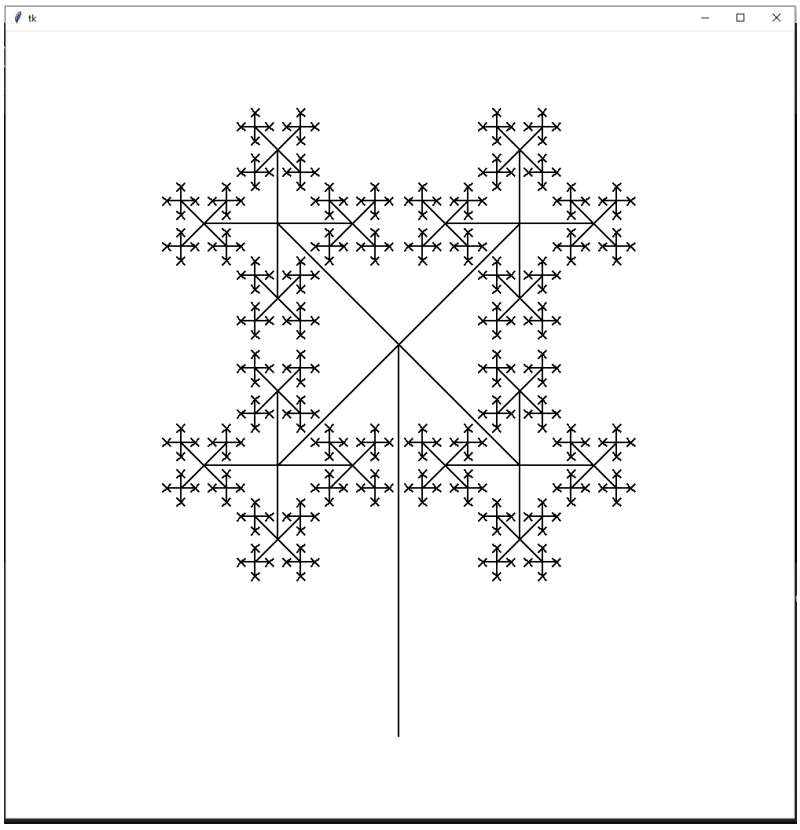
```
    # y1 - y koordināta galapunktam (nogriežņim)
    length = math.sqrt((x1 - x0)**2 + (y1 - y0)**2)

    if rekursijas_skaits == 0:
        return

    angle = math.atan2(y1 - y0, x1 - x0) # math.atan2() atgriež y/x arktangensu radiānos. Kur x un y ir punkta (x,y) koordinātas.

    draw_line(x0, y0, x1, y1)

    dx = math.cos(angle + math.pi / 4) * length / 2.3 # izmaiņa pēc x
    dy = math.sin(angle + math.pi / 4) * length / 2.3 # izmaiņa pēc y
    draw_crosses(x1, y1, x1 + dx, y1 + dy, rekursijas_skaits - 1)

    dx = math.cos(angle - math.pi / 4) * length / 2.3
    dy = math.sin(angle - math.pi / 4) * length / 2.3
    draw_crosses(x1, y1, x1 + dx, y1 + dy, rekursijas_skaits - 1)

    dx = math.cos(angle + math.pi * 3 / 4) * length / 2.3
    dy = math.sin(angle + math.pi * 3 / 4) * length / 2.3
    draw_crosses(x1, y1, x1 + dx, y1 + dy, rekursijas_skaits - 1)

    dx = math.cos(angle - math.pi * 3 / 4) * length / 2.3
    dy = math.sin(angle - math.pi * 3 / 4) * length / 2.3
    draw_crosses(x1, y1, x1 + dx, y1 + dy, rekursijas_skaits - 1)
# ------------------------------------------------------------
# Galvenā programmas daļa
# ------------------------------------------------------------
draw_crosses(500, 900, 500, 400, 5)
logs.mainloop()
```

Testa piemēri:

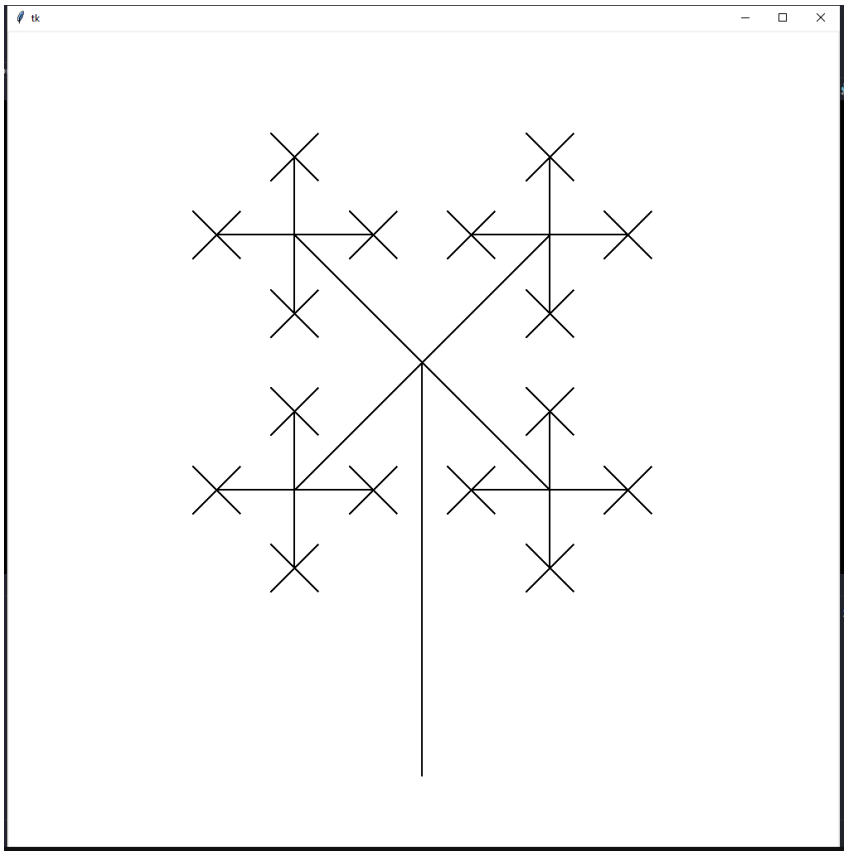
1) **rekursijas_skaits == 5**



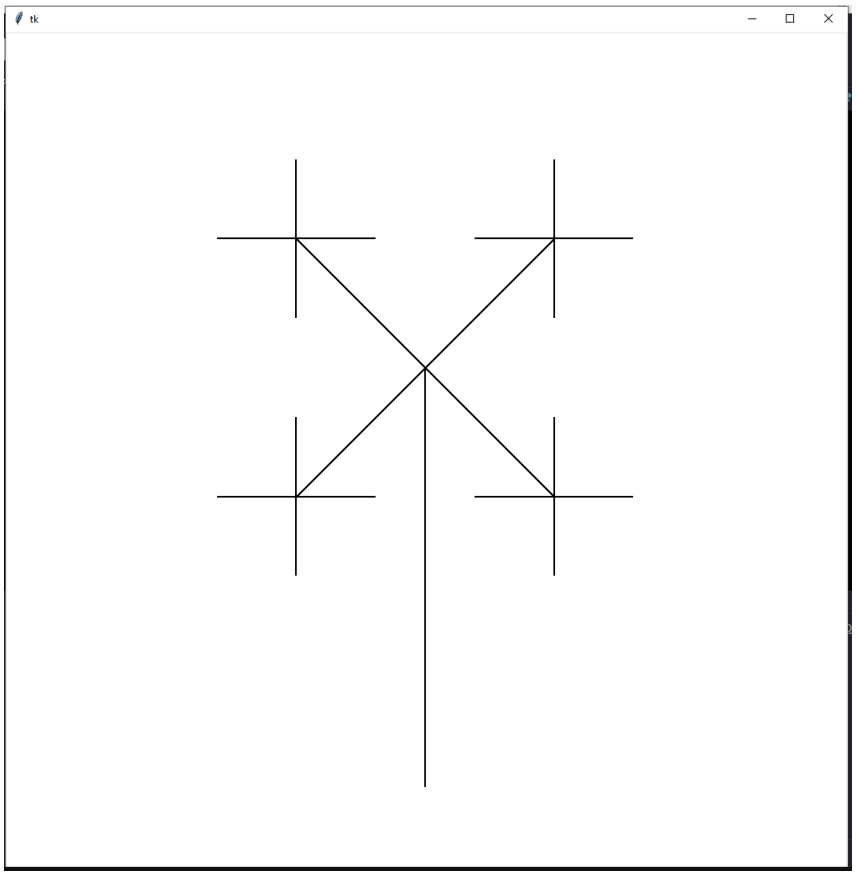
2) **rekursijas_skaits == 6**



3) **rekursijas_skaits == 4**

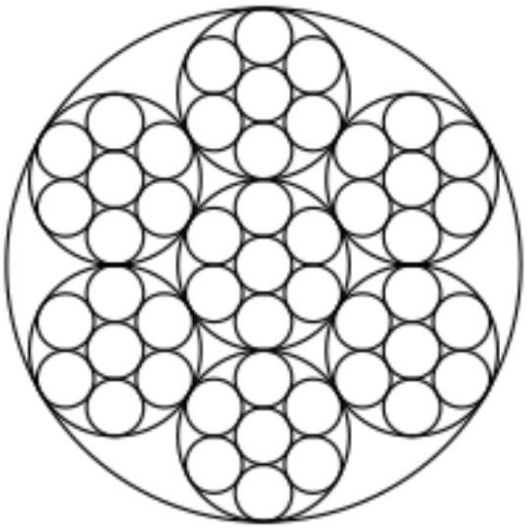


4) **rekursijas_skaits == 3**



PU2. uzdevums

Sastādīt programmu, kas izveido zemāk redzamo attēlu, izmantojot rekursiju.



Kods:

```
# Programmas nosaukums: Rekursija. Daudz riņķa līnijas.

# Papilduzdevums 2. (1MPR02_Vladislavs_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas izveido zemāk redzamo attēlu, izmantojot rekursiju.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0



import tkinter

import math



logs = tkinter.Tk()

logs.geometry("1000x1000")

W = 1000

H = 1000

canva = tkinter.Canvas(logs, width=W, height=H)

canva.configure(background='white')
```

```
canva.pack()


CENTRS_X = W // 2
CENTRS_Y = H // 2



def get_new_pos(x, y, lenkis, attalums):
    # Saņem x un y koordinātas, leņķi un attālumu un atgriež jauno pozīciju pēc dotā attāluma pārvietošanas dotajā leņķī
    # x - jaunais x
    # y - jaunais y
    # lenkis - leņķis grādos
    # attalums - attālums
    return x + attalums * math.cos(math.radians(lenkis)), y + attalums * math.sin(math.radians(lenkis))



def draw_circle(center_x, center_y, r):
    # Pēc dota centras koordinātas (center_x, center_y) un rādiusa (r) uzzīmē riņķa līniju.
    # center_x - riņķa līnijas centrs pēc x koordinātas
    # center_y - riņķa līnijas centrs pēc y koordinātas
    # r - riņķa līnijas rādiuss
    x1 = center_x - r
    x2 = center_x + r
    y1 = center_y - r
    y2 = center_y + r
    #print(x1, x2, y1, y2)
    canva.create_oval(x1, y1, x2, y2)
```

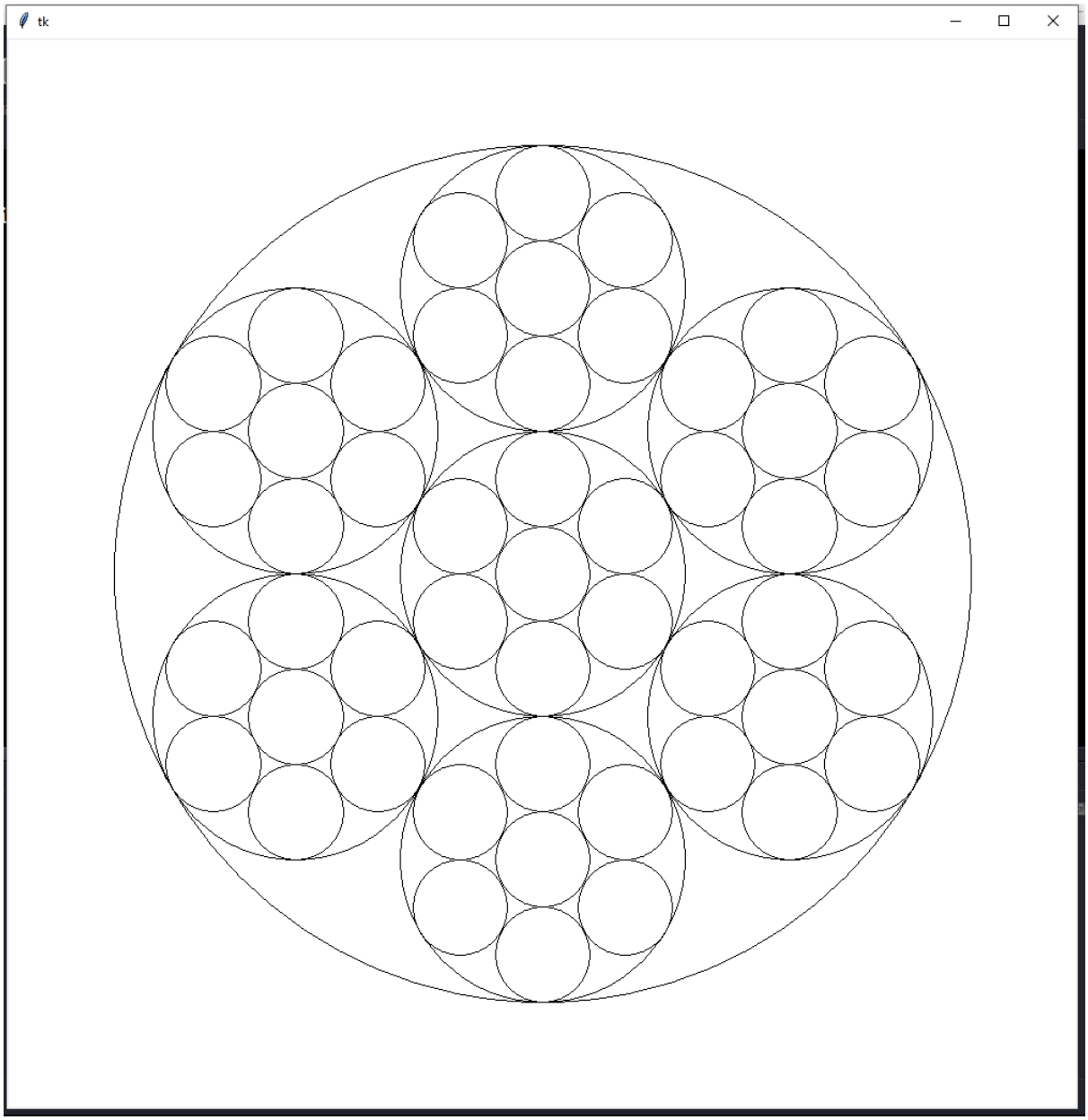
```
def recursion(center_x, center_y, r, rekursijas_skaits):
    # Tas saņem pašreizējās iterācijas centra koordinātas, rādiusu un dziļuma līmeni (rekursijas skaits).
    # Katrā dziļuma līmenī (rekursijas skaitam) tiek uzzīmēta riņķa līnijas pie dotajām centra koordinātām
    # un pēc tam funkcija tiek rekursīvi izsaukta ar sešām jaunām centra koordinātām,
    # kuras aprēķina, izmantojot funkciju get_new_pos.
    # center_x - riņķa līnijas centrs pēc x koordinātas
    # center_y - riņķa līnijas centrs pēc y koordinātas
    # r - riņķa līnijas rādiuss
    # rekursijas_skaits - rekursījas dziļums (cik reizes rekursija tiek īstenota)

    draw_circle(center_x, center_y, r)

    if rekursijas_skaits == 0:
        return
    attalums = r * 2 / 3
    jauns_radiuss = r / 3 # apzīmē mazāko apļu rādiusu, kas rekursīvi apvilkti ap lielākiem apļiem.
    lenkis = 30 # 30 grādi

    for t in range(6): # t netiek izmantota
        new_x, new_y = get_new_pos(center_x, center_y, lenkis, attalums)
        recursion(new_x, new_y, jauns_radiuss, rekursijas_skaits - 1)
        lenkis = lenkis + 60 # Katras jaunās centra koordinātas leņķis tiek nobīdīts par 60 grādiem no iepriekšējās, jo ap pašreizējo apli ir sešas riņķa līnijas

    recursion(center_x, center_y, jauns_radiuss, rekursijas_skaits - 1)

# ----------------------------------------------------------
# Galvenā programmas daļa
# ----------------------------------------------------------
```

recursion(CENTRS_X, CENTRS_Y, 400, 2)

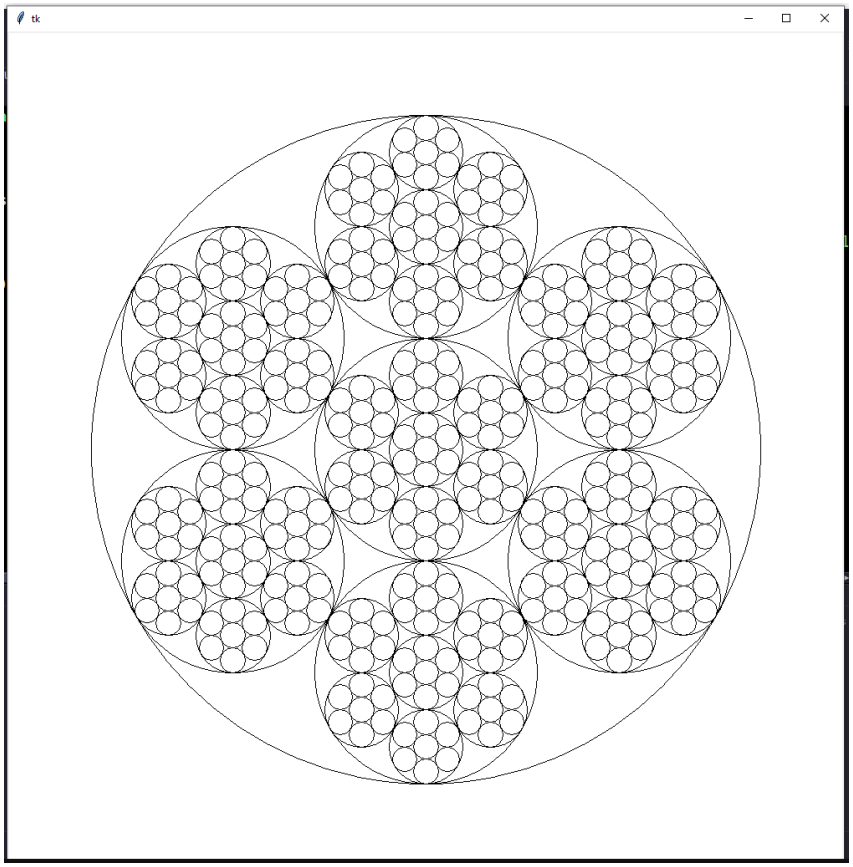
logs.mainloop()

Testa piemēri:

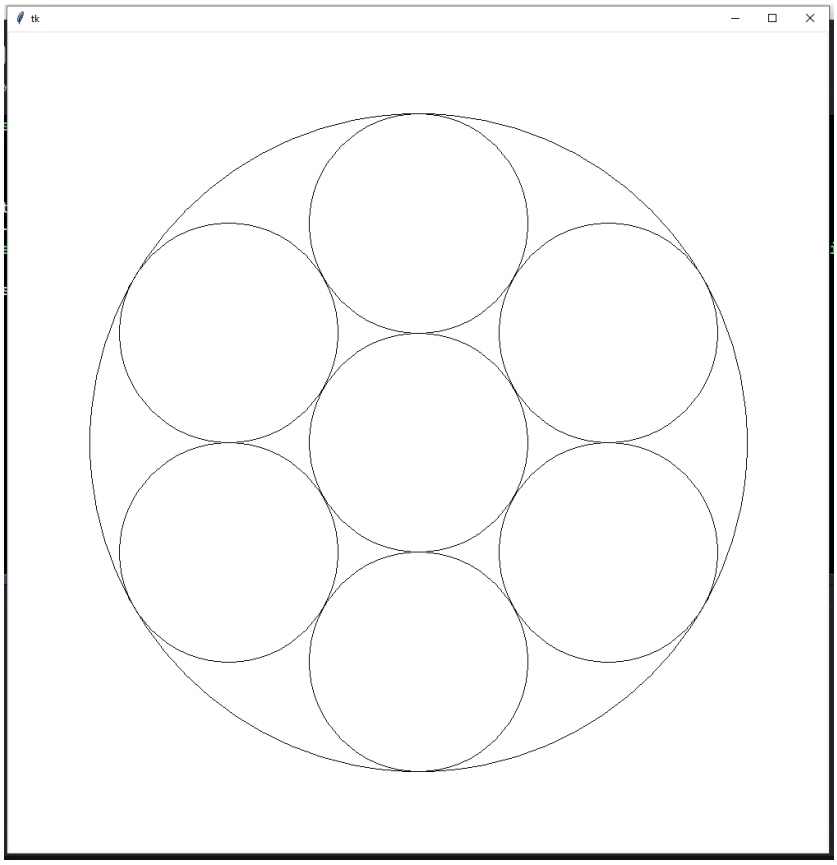
1) rekursijas_skaits == 2



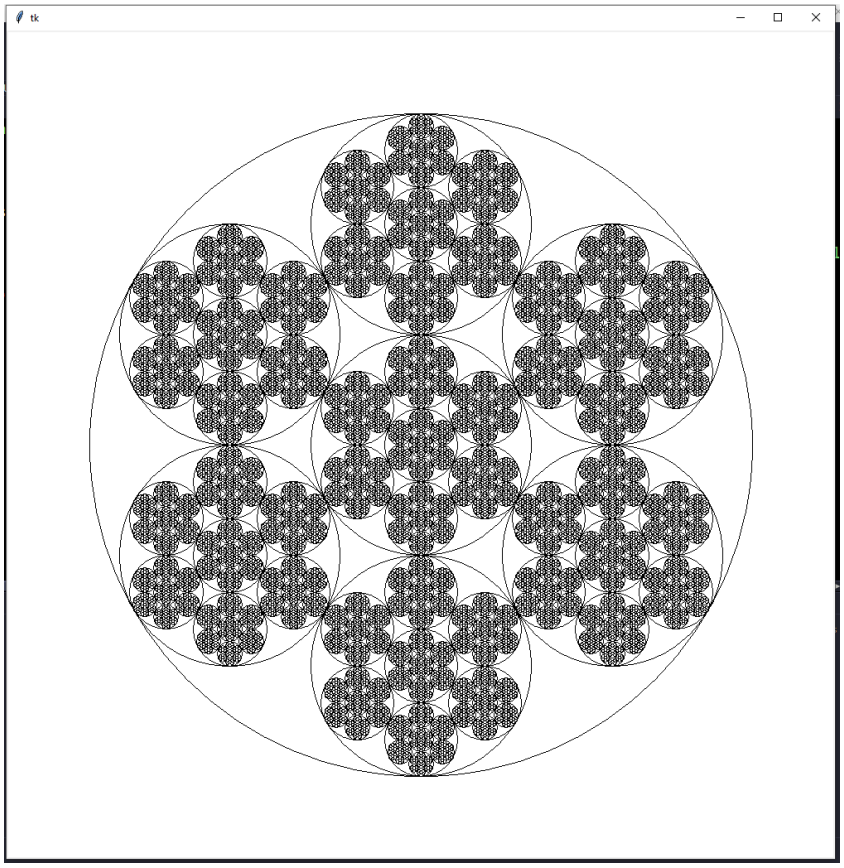
2) rekursijas_skaits == 2



3) rekursijas_skaits == 1



4) rekursijas_skaits == 5



5) rekursijas_skaits == 6