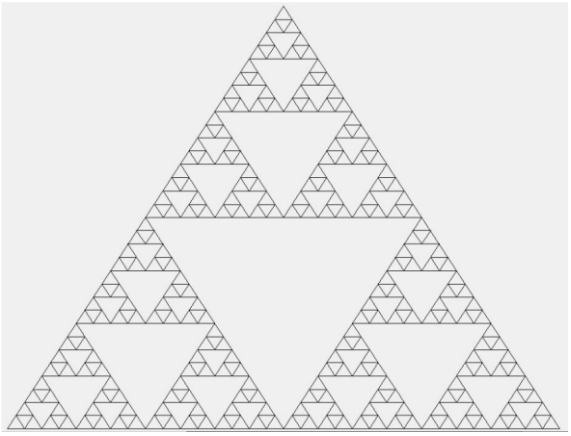
3. praktiskais darbs

1. uzdevums

Sastādīt programmu, kas izveido zemāk redzamo attēlu (Serpinska trijstūris), izmantojot rekursiju.



Kods:

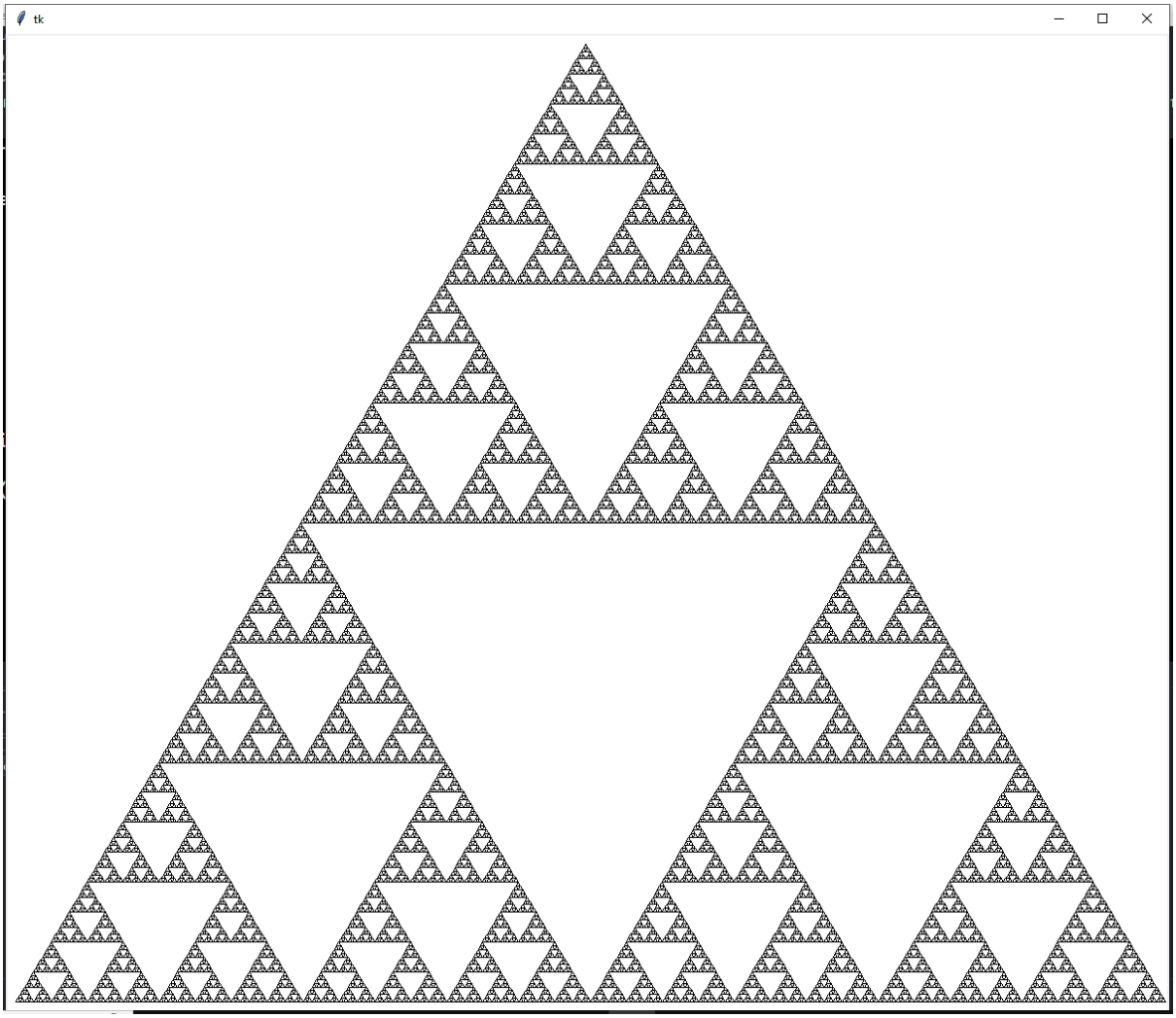
```
# Programmas nosaukums: Serpinska trijstūris.

# 1. uzdevums (1MPR03_Vladislavs_Babaņins)

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

# Programmas autors: Vladislavs Babaņins

# Versija 1.0



import tkinter



def draw_serpinskis(x1, y1, x2, y2, x3, y3):

    # Uzzīmē Serpinskā trijstūri izmantojot rekursiju

    # x1 - x koordināta kreisai apakšējai trijstūra virsotnei

    # y1 - y koordināta kreisai apakšējai trijstūra virsotnei

    # x2 - x koordināta labai apakšējai trijstūra virsotnei
```

```
    # y2 - y koordināta labai apakšējai trijstūra virsotnei
    # x3 - x koordināta augšējai centrālai trijstūra virsotnei
    # y3 - y koordināta augšējai centrālai trijstūra virsotnei

    area = abs((0.5) * (x1 * (y2 - y3) + x2 * (y3 - y1) + x3 * (y1 - y2))) # Trijstūra laukums

    if area < 10: # regulē rekursijas skaitu. Ja trijstūra laukums ir mazāks neka 10, tad stop
        kanva.create_polygon(x1, y1, x2, y2, x3, y3, fill='white', outline='black')
    else:
        x1x2 = (x1 + x2) / 2
        y1y2 = (y1 + y2) / 2
        x1x3 = (x1 + x3) / 2
        y1y3 = (y1 + y3) / 2
        x2x3 = (x2 + x3) / 2
        y2y3 = (y2 + y3) / 2

        draw_serpinskis(x1, y1, x1x2, y1y2, x1x3, y1y3)
        draw_serpinskis(x2, y2, x1x2, y1y2, x2x3, y2y3)
        draw_serpinskis(x3, y3, x1x3, y1y3, x2x3, y2y3)


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


logs = tkinter.Tk()

kanva = tkinter.Canvas(logs, width=1200, height=1200, bg='white')
kanva.pack()
```

```
x1 = 10

y1 = 1000

x2 = 1200

y2 = 1000

x3 = 1200

y3 = 10


draw_serpinskis(x1, y1, x2, y2, x3 / 2, y3)


logs.mainloop()
```

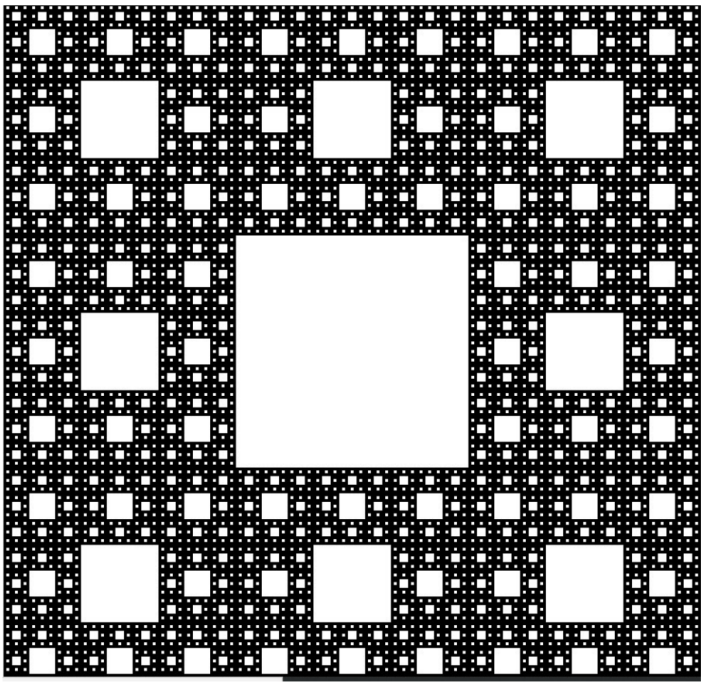
Testa piemēri:

1)



2. uzdevums

Sastādīt programmu, kas izveido zemāk redzamo attēlu (Serpinska paklājs), izmantojot rekursiju.



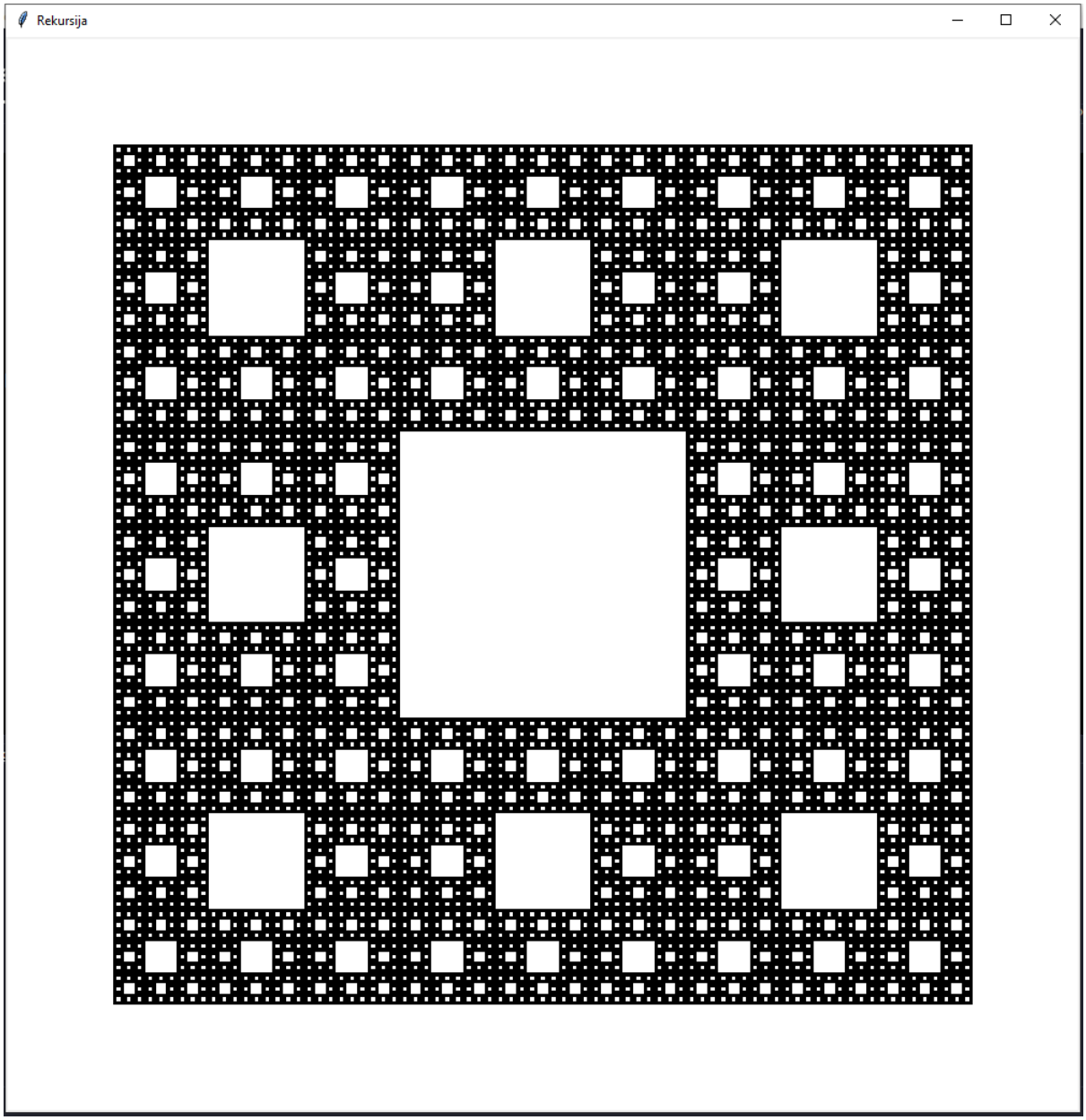
Kods:

```
# Programmas nosaukums: Serpinska paklājs.

# 2. uzdevums (1MPR03_Vladislavs_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas izveido Serpinska paklāju, izmantojot rekursiju.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0



import tkinter



def zimet_serpinska_paklaju(x, y, izmers):

    # Uzzīmē Serpinska paklāju

    # x - kreisa augšēja koordināta x Serpinska paklājam

    # y - kreisa augšēja koordināta y Serpinska paklājam

    if izmers < 5:
```

```
        kanva.create_rectangle(x, y, x + izmers, y + izmers, fill="black", outline="")
    else:
        maz_izmers = izmers / 3
        zimet_serpinska_paklaju(x, y, maz_izmers)
        zimet_serpinska_paklaju(x + maz_izmers, y, maz_izmers)
        zimet_serpinska_paklaju(x + 2 * maz_izmers, y, maz_izmers)
        zimet_serpinska_paklaju(x, y + maz_izmers, maz_izmers)
        zimet_serpinska_paklaju(x + 2 * maz_izmers, y + maz_izmers, maz_izmers)
        zimet_serpinska_paklaju(x, y + 2 * maz_izmers, maz_izmers)
        zimet_serpinska_paklaju(x + maz_izmers, y + 2 * maz_izmers, maz_izmers)
        zimet_serpinska_paklaju(x + 2 * maz_izmers, y + 2 * maz_izmers, maz_izmers)


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


logs = tkinter.Tk()
logs.geometry("1000x1000")
logs.title("Rekursija")
kanva = tkinter.Canvas(logs, width=1000, height=1000, bg="white")
kanva.pack()

zimet_serpinska_paklaju(100, 100, 800)

logs.mainloop()
```

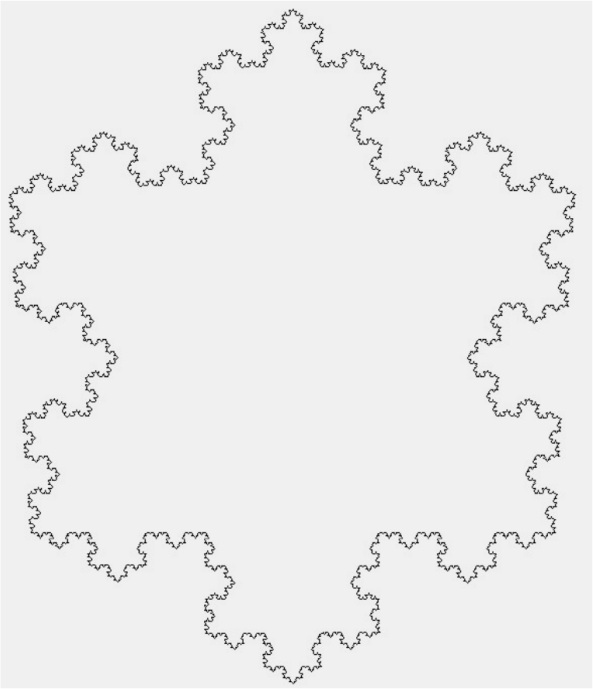
Testa piemēri:

1)



3. uzdevums

Sastādīt programmu, kas izveido zemāk redzamo attēlu (Koha zvaigzni), izmantojot rekursiju.



Kods:

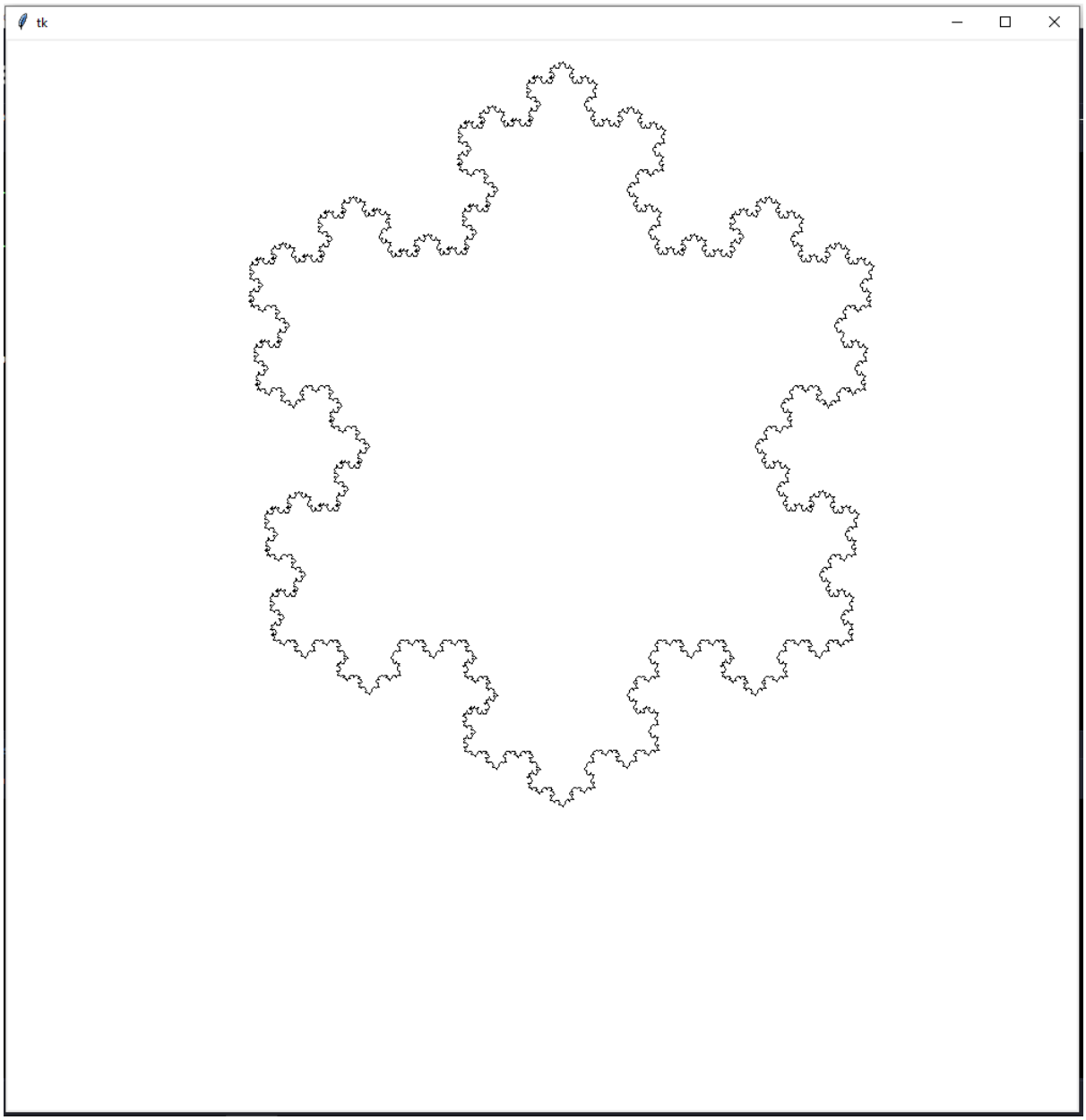
```
# Programmas nosaukums: Koha zvaigzne

# 3. uzdevums (1MPR03_Vladislavs_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas izveido Koha zvaigzni, izmantojot rekursiju.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0



import math

import tkinter



def zimet_koha_zvaigzni(x1, y1, x5, y5):

    # Rekursīvi uzzīme daļu no Koha zvaigznes

    # x1 - nogriežņa sākumpunkta x koordināta

    # y1 - nogriežņa sākumpunkta y koordināta
```

```
    # x5 - nogriežņa galapunkta x koordināta
    # y5 - nogriežņa galapunkta y koordināta
    if math.sqrt((x1 - x5)**2 + (y1 - y5)**2) < 4: # kad nogriežņa garums paliek mazāks par 4, tad pārtraucam rekursiju
        kanva.create_line(x1, y1, x5, y5)
    else:
        dx = x5 - x1
        dy = y5 - y1
        x2 = x1 + dx // 3
        y2 = y1 + dy // 3
        x3 = (x1 + x5) // 2 + math.sqrt(3) * (y1 - y5) // 6
        y3 = (y1 + y5) // 2 + math.sqrt(3) * (x5 - x1) // 6
        x4 = x1 + 2 * dx // 3
        y4 = y1 + 2 * dy // 3
        zimet_koha_zvaigzni(x1, y1, x2, y2)
        zimet_koha_zvaigzni(x2, y2, x3, y3)
        zimet_koha_zvaigzni(x3, y3, x4, y4)
        zimet_koha_zvaigzni(x4, y4, x5, y5)


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


logs = tkinter.Tk()
kanva = tkinter.Canvas(logs, bg="white", height=1000, width=1000)
kanva.pack()

zimet_koha_zvaigzni(250, 562, 792, 562)
zimet_koha_zvaigzni(792, 562, 521, 21)
```

```
zimet_koha_zvaigzni(521, 21, 250, 562)

logs.mainloop()
```

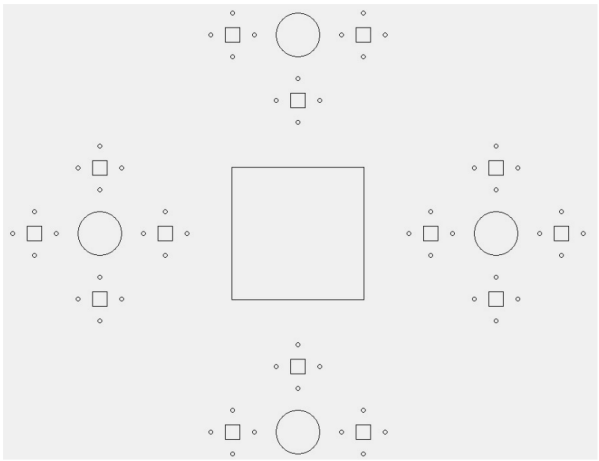
Testa piemēri:

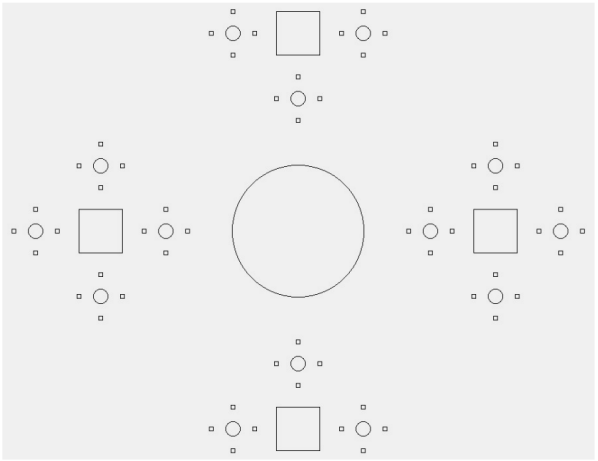
1)



4. uzdevums

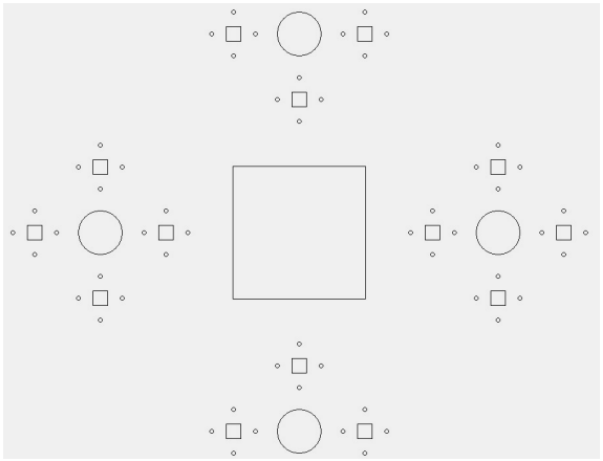
Sastādīt programmu, kas izveido zemāk redzamos attēlus, izmantojot rekursiju.





4.1. uzdevums

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



Kods:

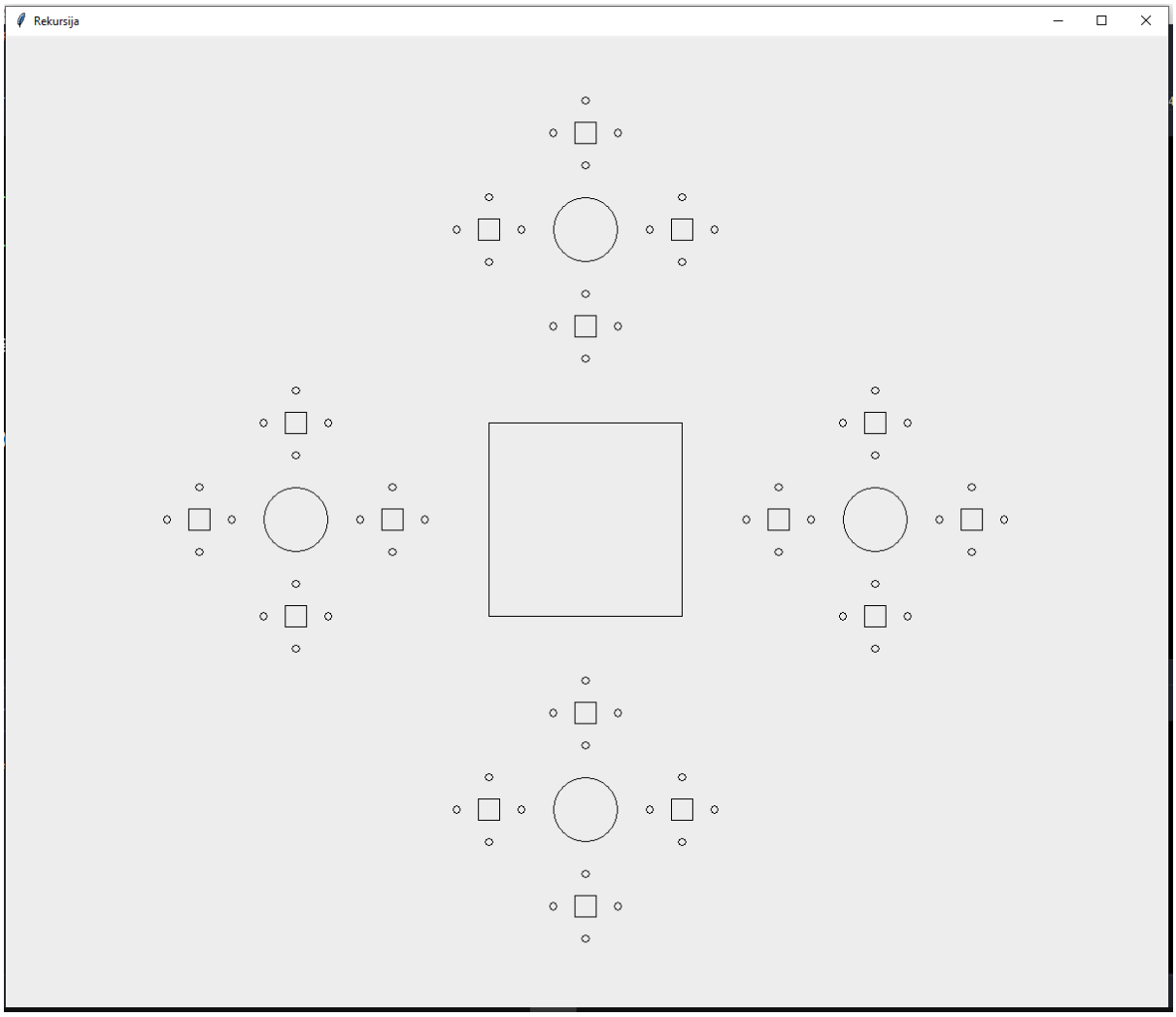
```
# Programmas nosaukums: Rekursija ar kvadrātiem un riņķa līnijām.

# 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


def rinka_linija(x, y, r):
    # Izveido riņķa līniju kurai centrs ir (x;y) un rādiuss ir r
    # x - x koordināta riņķa līnijas centram
    # y - y koordināta riņķa līnijas centram
    # r - riņķa līnijas rādiuss
    kanva.create_oval(x - r, y - r, x + r, y + r)


def kvadrats(x, y, r):
    # Izveido kvadrātu pēc dota centra koordinātam (x;y) un pēc r (puse no kvadrātas malas)
    # x - x koordināta kvadrātas centram
    # y - y koordināta kvadrātas centram
    # r - puse no kvadrātas malas (lai uzzimētu kvadrātu)
    kanva.create_rectangle(x - r, y - r, x + r, y + r)


def rekursija_1(x, y, r):
    # Uzzime kvadrātu un četras riņķa līnijas blakus pa visam pusem
    # x - x koordināta centram
    # y - y koordināta centram
    # r - rādiuss riņķa līnijai vai puse no kvadrātas malas
    kvadrats(x, y, r)
    if r > 5:
        rekursija_2(x + 3 * r, y, r / 3)
        rekursija_2(x - 3 * r, y, r / 3)
        rekursija_2(x, y + 3 * r, r / 3)
        rekursija_2(x, y - 3 * r, r / 3)
```

```
def rekursija_2(x, y, r):
    # Uzzime riņķa līniju un četrus kvadrātus blakus pa visam pusem
    # x - x koordināta centram
    # y - y koordināta centram
    # r - rādiuss riņķa līnijai vai puse no kvadrātas malas
    rinka_linija(x, y, r)
    if r > 5:
        rekursija_1(x + 3 * r, y, r / 3)
        rekursija_1(x - 3 * r, y, r / 3)
        rekursija_1(x, y + 3 * r, r / 3)
        rekursija_1(x, y - 3 * r, r / 3)



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



logs = tkinter.Tk()
kanva = tkinter.Canvas(logs, bg="#EDEDED", height=1200, width=1200)
kanva.pack()
logs.title("Rekursija")


rekursija_1(600, 500, 100)


logs.mainloop()
```

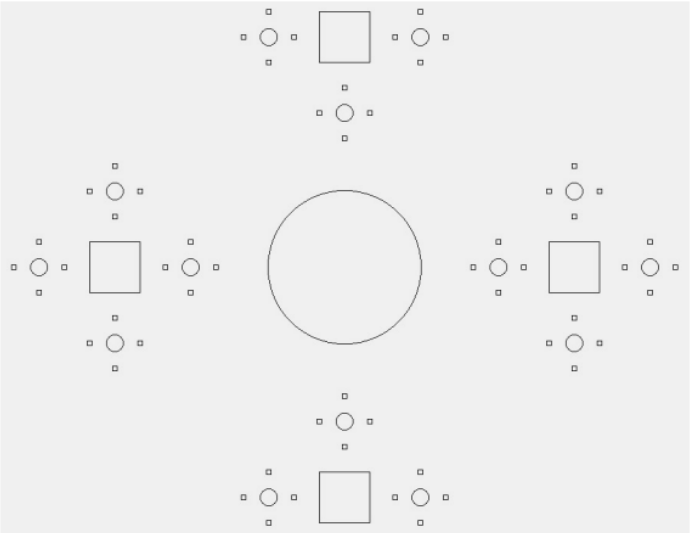
Testa piemēri:

1)



4.2. uzdevums

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



Kods:

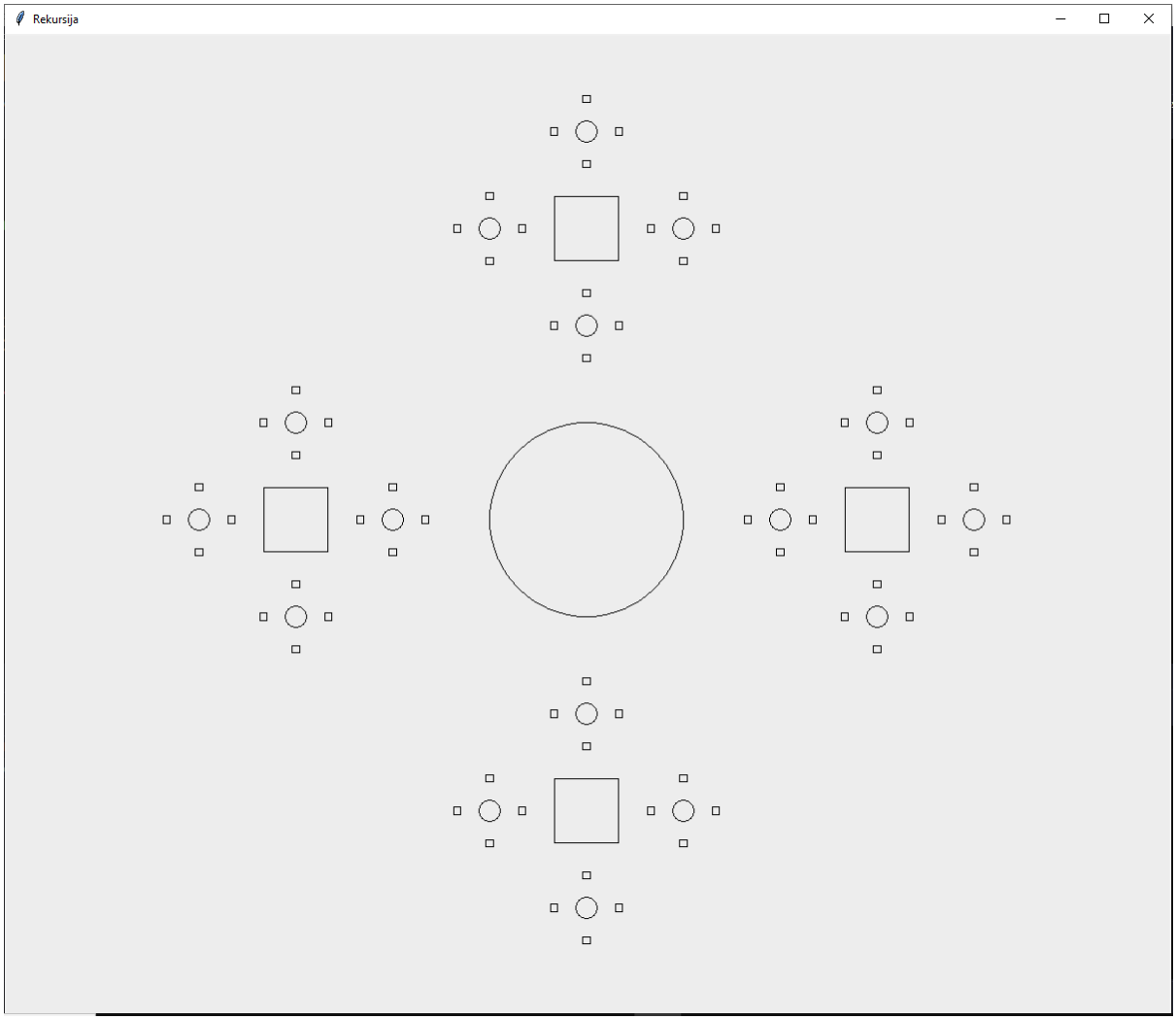
```
# Programmas nosaukums: Rekursija ar riņķa līnijām un kvadrātiem.

# 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



def rinka_linija(x, y, r):

    # Izveido riņķa līniju kurai centrs ir (x;y) un rādiuss ir r

    # x - x koordināta riņķa līnijas centram

    # y - y koordināta riņķa līnijas centram

    # r - riņķa līnijas rādiuss

    kanva.create_oval(x - r, y - r, x + r, y + r)
```

```
def kvadrats(x, y, r):
    # Izveido kvadrātu pēc dota centra koordinātam (x;y) un pēc r (puse no kvadrātas malas)
    # x - x koordināta kvadrātas centram
    # y - y koordināta kvadrātas centram
    # r - puse no kvadrātas malas (lai uzzimētu kvadrātu)
    kanva.create_rectangle(x - r, y - r, x + r, y + r)


def rekursija_1(x, y, r):
    # Uzzime kvadrātu un četras riņķa līnijas blakus pa visam pusem
    # x - x koordināta centram
    # y - y koordināta centram
    # r - rādiuss riņķa līnijai vai puse no kvadrātas malas
    kvadrats(x, y, r)
    if r > 5:
        rekursija_2(x + 3 * r, y, r / 3)
        rekursija_2(x - 3 * r, y, r / 3)
        rekursija_2(x, y + 3 * r, r / 3)
        rekursija_2(x, y - 3 * r, r / 3)


def rekursija_2(x, y, r):
    # Uzzīmē riņķa līniju un četrus kvadrātus blakus pa visam pusēm
    # x - x koordināta centram
    # y - y koordināta centram
    # r - rādiuss riņķa līnijai vai puse no kvadrātas malas
    rinka_linija(x, y, r)
    if r > 5:
        rekursija_1(x + 3 * r, y, r / 3)
```

```
        rekursija_1(x - 3 * r, y, r / 3)
        rekursija_1(x, y + 3 * r, r / 3)
        rekursija_1(x, y - 3 * r, r / 3)



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



logs = tkinter.Tk()
kanva = tkinter.Canvas(logs, bg="#EDEDED", height=1200, width=1200)
kanva.pack()
logs.title("Rekursija")

rekursija_2(600, 500, 100)

logs.mainloop()
```

Testa piemēri:

1)



5. uzdevums

Pērlīšu vēršanas uzdevums. Zināms, ka vienai zilai pērlītei var pievienot tieši 2 sarkanas, 1 zilu un 3 oranžas pērlītes; vienai sarkanai pērlītei var pievienot tieši 3 sarkanas, 2 zilas un 1 oranžu pērlīti; vienai oranžai pērlītei var pievienot tieši 2 sarkanas, 3 zilas un 1 oranžu pērlīti; pērlītes tiek izkārtotas pa rindiņām, un katrā nākamajā rindiņā esošās pērlītes ir visu iepriekšējā rindiņā esošo pērlīšu "pēcteči". Sastādīt programmu, kas noskaidro cik un kādas krāsas pērlītes ir **N**-tajā rindiņā. Skaitli **N** ievada lietotājs.

Kods:

```
# Programmas nosaukums: Pērlīšu vēršanas uzdevums.

# 5. uzdevums (1MPR03_Vladislavs_Babaņins)

# Uzdevuma formulējums: Zināms, ka vienai zilai pērlītei var pievienot tieši 2 sarkanas, 1 zilu un 3 oranžas pērlītes;

# vienai sarkanai pērlītei var pievienot tieši 3 sarkanas, 2 zilas un 1 oranžu pērlīti;

# vienai oranžai pērlītei var pievienot tieši 2 sarkanas, 3 zilas un 1 oranžu pērlīti;

# pērlītes tiek izkārtotas pa rindiņām, un katrā nākamajā rindiņā esošās pērlītes ir visu iepriekšējā rindiņā esošo pērlīšu "pēcteči".

# Sastādīt programmu, kas noskaidro cik un kādas krāsas pērlītes ir N-tajā rindiņā. Skaitli N ievada lietotājs.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0



def is_natural(n):

    # Pārbauda vai simbolu virkne ir naturāls skaitlis vai nav

    # Ja ir naturāls skaitlis, tad True. Ja nav tad False.

    # n - simbolu virkne, kuru pārbauda.

    if str(n).isdigit() and float(n) == int(n) and int(n) > 0:

        return True

    else:

        return False



def zila(N, krasa):
```

```
    # Skaita cik ir zīlas krāsas pērlītes N rindā
    # N - N rindā
    # krasa - "z" - zīla, "s" - sarkana, "o" - oranža
    if N == 1:
        if krasa == "z":
            return 1
        else:
            return 0
    else:
        return 2 * sarkana((N - 1), krasa) + zila((N - 1), krasa) + 3 * oranzs((N - 1), krasa)


def sarkana(N, krasa):
    # Skaita cik ir sarkanas krāsas pērlītes N rindā
    # N - N rindā
    # krasa - "z" - zīla, "s" - sarkana, "o" - oranža
    if N == 1:
        if krasa == "s":
            return 1
        else:
            return 0
    else:
        return 3 * sarkana((N - 1), krasa) + 2 * zila((N - 1), krasa) + 2 * oranzs((N - 1), krasa)


def oranzs(N, krasa):
    # Skaita cik ir oranžas krāsas pērlītes N rindā
    # N - N rindā
    # krasa - "z" - zīla, "s" - sarkana, "o" - oranža
    if N == 1:
        if krasa == "o":
```

```
            return 1
        else:
            return 0

    else:
        return sarkana((N - 1), krasa) + 3 * zila((N - 1), krasa) + oranzs((N - 1), krasa)


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


krasa = input("Ievadiet pirmā pērliša krāsu!\n'z' (zilā), 's' (sarkanā), 'o' (oranžā) ==> ")

while krasa != "s" and krasa != "z" and krasa != "o":
    krasa = input("Tika ievadīti nekorekti dati.\nIevadiet pirmā pērliša krāsu!\n 'z' (zilā), 's' (sarkanā), 'o' (oranžā) ==> ")

N = input("Ievadiet rindas numuru ==> ")

while is_natural(N) == False:
    N = input("Tika ievadīti nekorekti dati.\nIevadiet rindas numuru ==> ")

N = int(N)

# Izvada pērlīšu skaitu katra krāsā N rindā
print("Sarkanas pērlītes:", sarkana(N, krasa))
print("Zilās pērlītes:", zila(N, krasa))
print("Oranžas pērlītes:", oranzs(N, krasa))
```

Testa piemēri:

1)

```
Ievadiet pirmā pērliša krāsu!
'z' (zilā), 's' (sarkanā), 'o' (oranžā) ==> z
Ievadiet rindas numuru ==> 1
Sarkanas pērlītes: 0
Zilās pērlītes: 1
Oranžas pērlītes: 0
```

2)

```
Ievadiet pirmā pērliša krāsu!
'z' (zilā), 's' (sarkanā), 'o' (oranžā) ==> s
Ievadiet rindas numuru ==> 1
Sarkanas pērlītes: 1
Zilās pērlītes: 0
Oranžas pērlītes: 0
```

3)

```
Ievadiet pirmā pērliša krāsu!
'z' (zilā), 's' (sarkanā), 'o' (oranžā) ==> o
Ievadiet rindas numuru ==> 1
Sarkanas pērlītes: 0
Zilās pērlītes: 0
Oranžas pērlītes: 1
```

4)

```
Ievadiet pirmā pērliša krāsu!
'z' (zilā), 's' (sarkanā), 'o' (oranžā) ==> z
Ievadiet rindas numuru ==> 2
Sarkanas pērlītes: 2
Zilās pērlītes: 1
Oranžas pērlītes: 3
```

5)

```
Ievadiet pirmā pērliša krāsu!
'z' (zilā), 's' (sarkanā), 'o' (oranžā) ==> s
Ievadiet rindas numuru ==> 2
Sarkanas pērlītes: 3
Zilās pērlītes: 2
Oranžas pērlītes: 1
```

6)

```
Ievadiet pirmā pērliša krāsu!
'z' (zilā), 's' (sarkanā), 'o' (oranžā) ==> o
Ievadiet rindas numuru ==> 2
Sarkanas pērlītes: 2
Zilās pērlītes: 3
Oranžas pērlītes: 1
```

7)

```
Ievadiet pirmā pērliša krāsu!
'z' (zilā), 's' (sarkanā), 'o' (oranžā) ==> z
Ievadiet rindas numuru ==> 3
Sarkanas pērlītes: 14
Zilās pērlītes: 14
Oranžas pērlītes: 8
```

8)

```
Ievadiet pirmā pērliša krāsu!
'z' (zilā), 's' (sarkanā), 'o' (oranžā) ==> s
Ievadiet rindas numuru ==> 3
Sarkanas pērlītes: 15
Zilās pērlītes: 11
Oranžas pērlītes: 10
```

9)

```
Ievadiet pirmā pērliša krāsu!
'z' (zilā), 's' (sarkanā), 'o' (oranžā) ==> o
Ievadiet rindas numuru ==> 3
Sarkanas pērlītes: 14
Zilās pērlītes: 10
Oranžas pērlītes: 12
```

10)

```
Ievadiet pirmā pērliša krāsu!
'z' (zilā), 's' (sarkanā), 'o' (oranžā) ==> z
Ievadiet rindas numuru ==> 4
Sarkanas pērlītes: 86
Zilās pērlītes: 66
Oranžas pērlītes: 64
```

11)

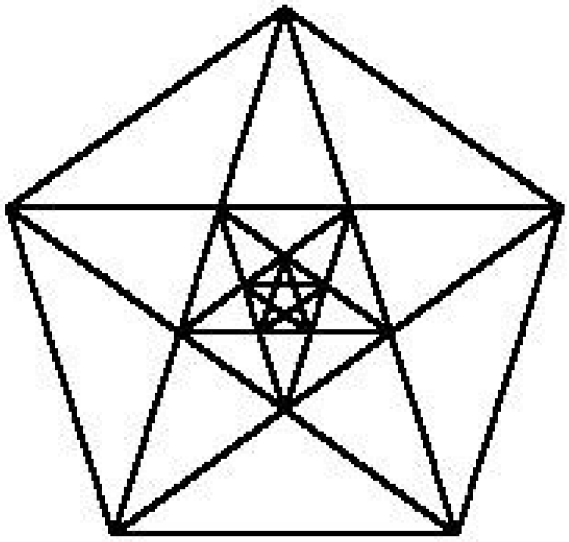
```
Ievadiet pirmā pērliša krāsu!
'z' (zilā), 's' (sarkanā), 'o' (oranžā) ==> Olbanca
Tika ievadīti nekorekti dati.
Ievadiet pirmā pērliša krāsu!
 'z' (zilā), 's' (sarkanā), 'o' (oranžā) ==> Pumca
Tika ievadīti nekorekti dati.
Ievadiet pirmā pērliša krāsu!
 'z' (zilā), 's' (sarkanā), 'o' (oranžā) ==> s
Ievadiet rindas numuru ==> rinda numur'5
Tika ievadīti nekorekti dati.
Ievadiet rindas numuru ==> pieci
Tika ievadīti nekorekti dati.
Ievadiet rindas numuru ==> 5
Sarkanas pērlītes: 519
Zilās pērlītes: 419
Oranžas pērlītes: 358
```

PU1.

Sastādīt programmu, kas izveido zemāk redzamos attēlus, izmantojot rekursiju.

PU1.1.

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



Kods:

```
# Programmas nosaukums: Rekursija ar piecstūrim.

# PU1.1 uzdevums (1MPR03_Vladislavs_Babaņins)

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

# Programmas autors: Vladislavs Babaņins

# Versija 1.0



import math

import tkinter



def zimet_piecsturi_ar_diagonalem(x0, y0, R):

    # uzzīmē regulāro piecstūrī (nav rotēts)

    # x0 - regulāra piecstūra centra x koordināta
```

```
# y0 - regulāra piecstūra centra y koordināta
# R - regulāra piecstūra rādiuss

# t - regulāra piecstūra vienas malas garums (nepieciešams, lai izrēķinātu koordinātas)
# h - regulāra piecstūra augstuma garums (nepieciešams, lai izrēķinātu koordinātas)

t = R * math.sqrt((5 - math.sqrt(5)) / 2) # R * 1.17557
h = (math.tan(math.radians(72)) / 2) * t # 1.539 * t
k = t * math.sin(math.radians(54)) # k = t*sin(54 degree) (palīgnogriežnis)
p = t * math.cos(math.radians(54)) # p = t*cos(54 degree) (palīgnogriežnis)

# piecstūra koordinātas
x1 = x0 - t / 2
y1 = y0 - R + h

x2 = x0 - k
y2 = y0 - R + p

x3 = x0
y3 = y0 - R

x4 = x0 + k
y4 = y0 - R + p

x5 = x0 + t / 2
y5 = y0 - R + h

# pati piecstūra uzzīmēšanā
kanva.create_line(x1, y1, x2, y2)
kanva.create_line(x2, y2, x3, y3)
kanva.create_line(x3, y3, x4, y4)
```

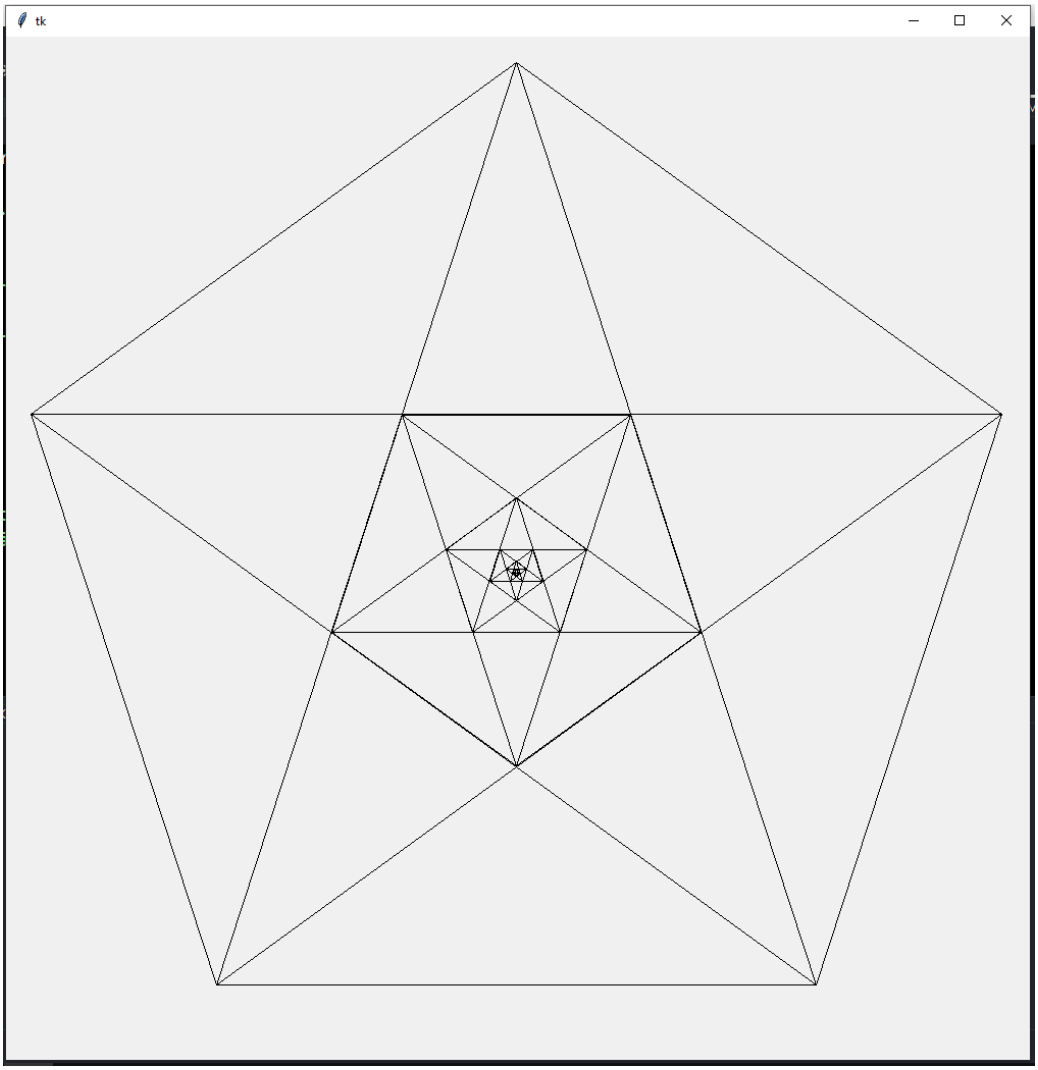
```
    kanva.create_line(x4, y4, x5, y5)
    kanva.create_line(x5, y5, x1, y1)

    # diagonāles uzzīmēšanā
    kanva.create_line(x1, y1, x3, y3)
    kanva.create_line(x1, y1, x4, y4)

    kanva.create_line(x2, y2, x4, y4)
    kanva.create_line(x2, y2, x5, y5)

    kanva.create_line(x3, y3, x5, y5)


def zimet_piecsturi_ar_diagonalem_rotets(x0, y0, R):
    # uzzīme regulāro piecsturi (ir rotēts)
    # x0 - regulāra piecstūra centra x koordināta
    # y0 - regulāra piecstūra centra y koordināta
    # R - regulāra piecstūra rādiuss

    # t - regulāra piecstūra vienas malas garums (nepieciešams, lai izrēķinātu koordinātas)
    # h - regulāra piecstūra augstuma garums (nepieciešams, lai izrēķinātu koordinātas)

    t = R * math.sqrt((5 - math.sqrt(5)) / 2) # t aptuvēni = R * 1.17557
    h = (math.tan(math.radians(72)) / 2) * t # h aptuvēni = 1.539 * t
    k = t * math.sin(math.radians(54)) # k = t*sin(54 degree) (palīgnogriežnis)
    p = t * math.cos(math.radians(54)) # p = t*cos(54 degree) (palīgnogriežnis)

    # piecstūra koordinātas
    x1 = x0
    y1 = y0 + R
```

```
    x2 = x0 - k
    y2 = y0 + R - p

    x3 = x0 - t / 2
    y3 = y0 + R - h

    x4 = x0 + t / 2
    y4 = y0 + R - h

    x5 = x0 + k
    y5 = y0 + R - p

    # pati piecstūra uzzīmēšanā
    kanva.create_line(x1, y1, x2, y2)
    kanva.create_line(x2, y2, x3, y3)
    kanva.create_line(x3, y3, x4, y4)
    kanva.create_line(x4, y4, x5, y5)
    kanva.create_line(x5, y5, x1, y1)

    # diagonāles uzzīmēšanā
    kanva.create_line(x1, y1, x3, y3)
    kanva.create_line(x1, y1, x4, y4)

    kanva.create_line(x2, y2, x4, y4)
    kanva.create_line(x2, y2, x5, y5)

    kanva.create_line(x3, y3, x5, y5)



def zimet_piecsturi_rekursivi(x0, y0, R):
    # zime rekursīvi piecstūrus ar diagonālem izsaucot divas citas funkcijas
```

```
    # rekursija beigās kad R <= 10
    if R > 10:

        zimet_piecsturi_ar_diagonalem(x0, y0, R)
        zimet_piecsturi_ar_diagonalem_rotets(x0, y0, R * math.pi / 8.25)
        r = R * math.pi / 8.25
        zimet_piecsturi_rekursivi(x0, y0, r * math.pi / 8.25)



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



logs = tkinter.Tk()
kanva = tkinter.Canvas(logs, width=1000, height=1000)
kanva.pack()

x0 = 500 # vislielākā (pirmā) piecstūra centra x koordināta
y0 = 525 # vislielākā (pirmā) piecstūra centra y koordināta
R = 500 # vislielākā (pirmā) piecstūra rādiusa vērtība

zimet_piecsturi_rekursivi(x0, y0, R)

logs.mainloop()
```

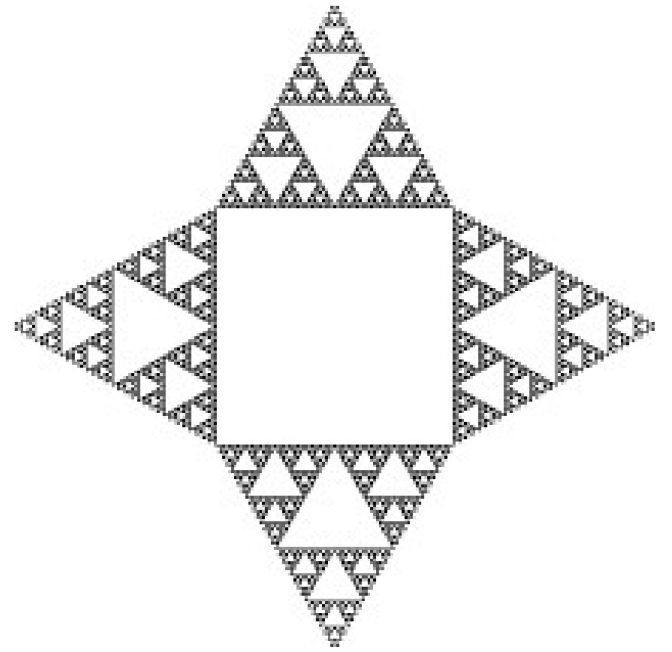
Testa piemēri:

1)



PU1.2.

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



Kods:

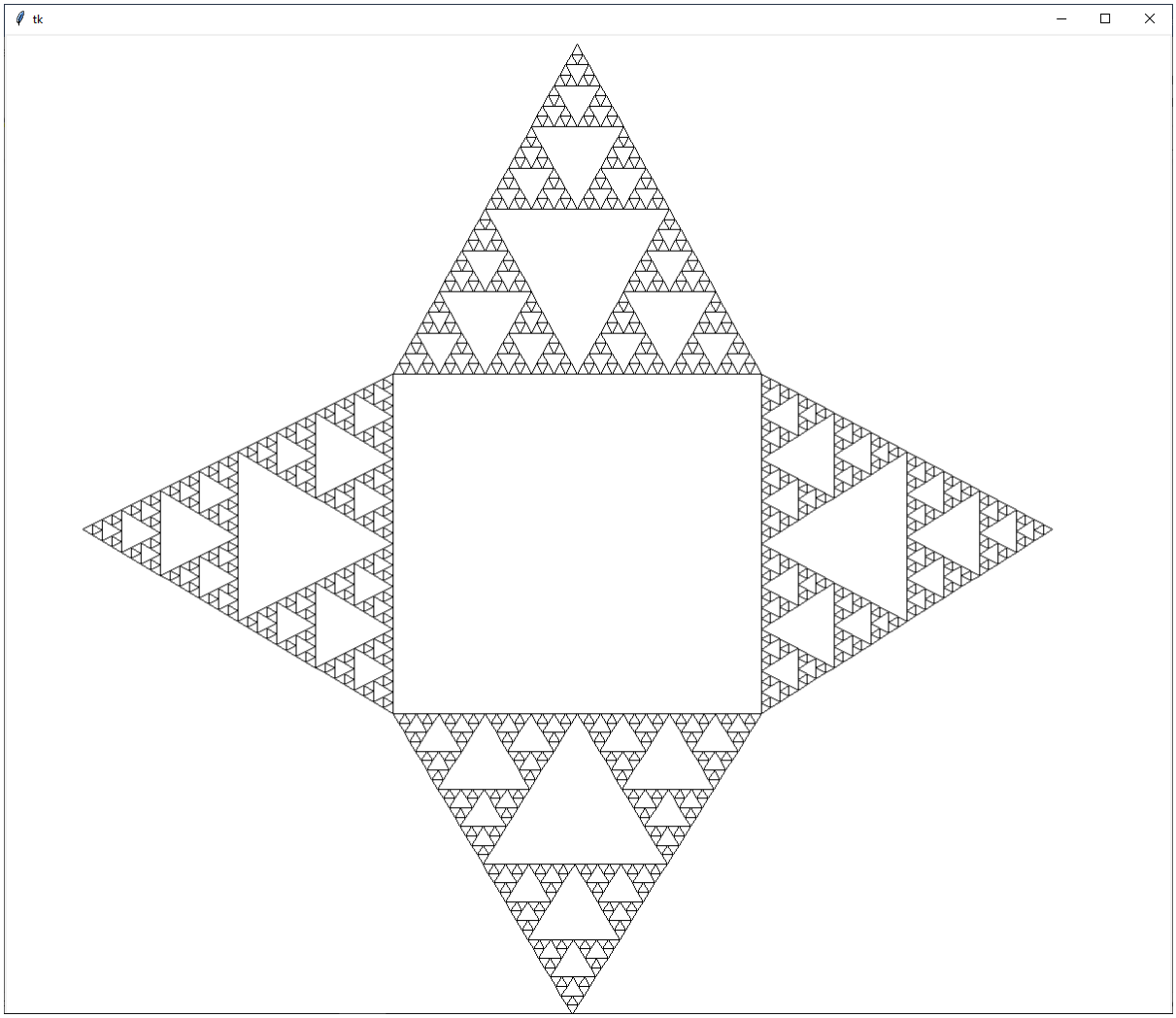
```
# Programmas nosaukums: Rekursija četri Serpinska trijstūri.

# PU1.2 uzdevums (1MPR03_Vladislavs_Babaņins)

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

# Programmas autors: Vladislavs Babaņins

# Versija 1.0



import tkinter



def draw_serpinskis(x1, y1, x2, y2, x3, y3, rekursijas_skaits):

    # uzzīmē vienu Serpinska trijstūrī izmantojot rekursiju

    # x1 - vislielākā trijstūra koordināta x vienai virsotnei

    # y1 - vislielākā trijstūra koordināta y vienai virsotnei

    # x2 - vislielākā trijstūra koordināta x otrai virsotnei
```

```
    # y2 - vislielākā trijstūra koordināta y otrai virsotnei
    # x3 - vislielākā trijstūra koordināta x trešai virsotnei
    # y3 - vislielākā trijstūra koordināta y trešai virsotnei
    # rekursijas_skaits - rekursijas skaits (cik līmeņus ir jāuzzīmē Serpinska trijstūri)

    if rekursijas_skaits == 0:
        kanva.create_polygon(x1, y1, x2, y2, x3, y3, fill='white', outline='black')
    else:
        x1x2 = (x1 + x2) / 2
        y1y2 = (y1 + y2) / 2
        x1x3 = (x1 + x3) / 2
        y1y3 = (y1 + y3) / 2
        x2x3 = (x2 + x3) / 2
        y2y3 = (y2 + y3) / 2

        draw_serpinskis(x1, y1, x1x2, y1y2, x1x3, y1y3, rekursijas_skaits - 1)
        draw_serpinskis(x2, y2, x1x2, y1y2, x2x3, y2y3, rekursijas_skaits - 1)
        draw_serpinskis(x3, y3, x1x3, y1y3, x2x3, y2y3, rekursijas_skaits - 1)



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



logs = tkinter.Tk()

kanva = tkinter.Canvas(logs, width=1200, height=1300, bg='white')
kanva.pack()

x1 = 400
```

```
y1 = 350


x2 = 780
y2 = 350


x3 = 1180
y3 = 10


rekursijas_skaits = 5


draw_serpinskis(x1, y1, x2, y2, x3 / 2, y3, rekursijas_skaits)
draw_serpinskis(x1, y1, x2 - x1 + 20, y1 + y2, x3 - 1100, y3 + 500, rekursijas_skaits)
draw_serpinskis(x1 + 380, y1 + 350, x2 - x1 + 20, y1 + y2, x3 - 595, y3 + 1000, rekursijas_skaits)
draw_serpinskis(x1 + 380, y1, x2 - x1 + 400, y1 + y2, x3 - 100, y3 + 500, rekursijas_skaits)


logs.mainloop()
```

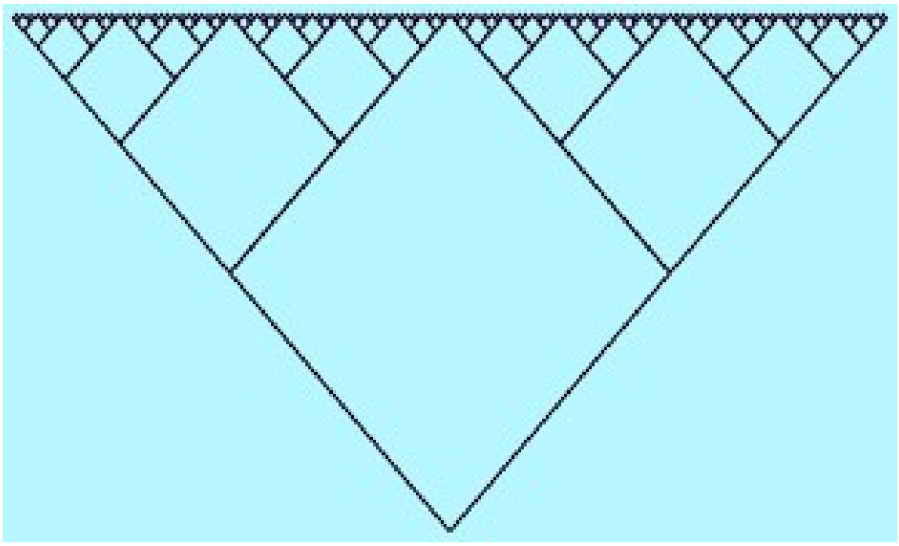
Testa piemēri:

1)



PU1.3.

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



Kods:

Programmas nosaukums: Rekursija. V-koks.

PU1.3 uzdevums (1MPR03_Vladislavs_Babaņins)

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

Programmas autors: Vladislavs Babaņins

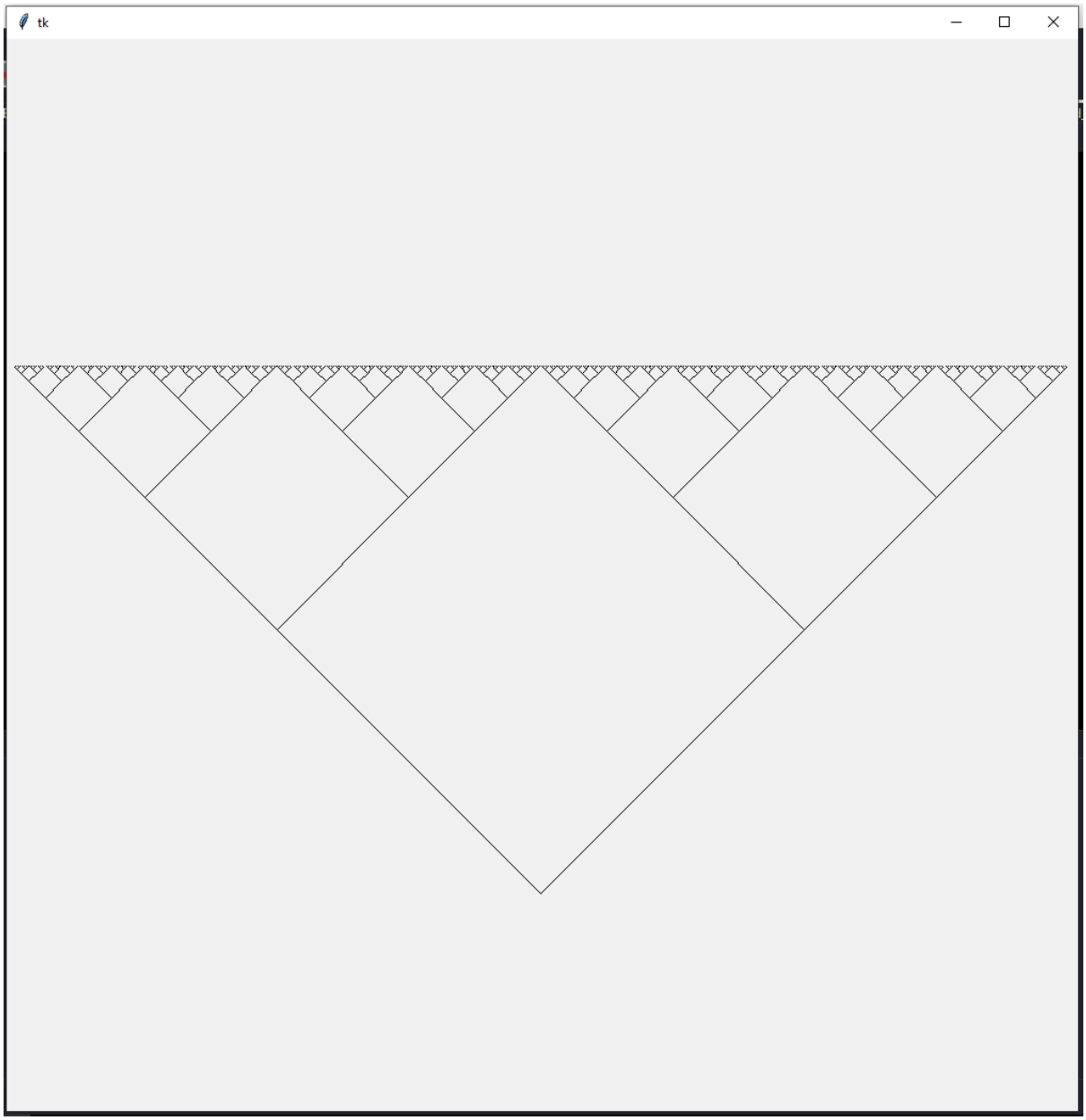
Versija 1.0

```
import math

import tkinter



def zimet(x, y, garums):

    # uzzīmē rekursīvi "koku"

    # x - pirmā nogriežņa x koordināta

    # y - pirmā nogriežņa y koordināta

    # garums - pirmā nogriežņa garums (pēc tā tiek izrēķinātas koordinātas nepieciešamas, lai uzzīmēt nogriezni)
```

```
    lx = x + garums * math.cos(math.pi * 7 / 4) # -1*math.pi / 4 (-45 gradi)
    ly = y + garums * math.sin(math.pi * 7 / 4) # -1*math.pi / 4 (-45 gradi)

    lenkis = -1 * math.pi / 2 + math.pi * 7 / 4
    rx = x + garums * math.cos(lenkis)
    ry = y + garums * math.sin(lenkis)

    kanva.create_line(x, y, lx, ly)
    kanva.create_line(x, y, rx, ry)

    if garums > 1: # Stop kad garums nogrieznim ir mazāks nekā 1
        zimet(lx, ly, garums / 2)
        zimet(rx, ry, garums / 2)



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



logs = tkinter.Tk()
kanva = tkinter.Canvas(logs, width=1000, height=1000)
kanva.pack()

zimet(500, 800, 350)

logs.mainloop()
```

Testa piemēri

1)



PU1.4.

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

Kods:

```
# Programmas nosaukums: Rekursija. Kvadrāti kvadrātos.

# PU1.4 uzdevums (1MPR03_Vladislavs_Babaņins)

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

# Programmas autors: Vladislavs Babaņins

# Versija 1.0



import math

import tkinter



def zimet_rekursivi(x0, y0, x2, y2, x1, y1, x3, y3):

    # Uzzīmē rekursīvi kvadrātus noteiktā secība

    # x0 - kreisā apakšēja stūra koordināta pēc x (runājot par 1. iterācijas kvadrātu)

    # y0 - kreisā apakšēja stūra koordināta pēc y (runājot par 1. iterācijas kvadrātu)



    # x2 - laba augšēja stūra koordināta pēc x (runājot par 1. iterācijas kvadrātu)

    # y2 - laba augšēja stūra koordināta pēc y (runājot par 1. iterācijas kvadrātu)
```

```
    # x1 - kreisā augšēja stūra koordināta pēc x (runājot par 1. iterācijas kvadrātu)
    # y1 - kreisā augšēja stūra koordināta pēc y (runājot par 1. iterācijas kvadrātu)

    # x3 - laba apakšēja stūra koordināta pēc x (runājot par 1. iterācijas kvadrātu)
    # y3 - laba apakšēja stūra koordināta pēc x (runājot par 1. iterācijas kvadrātu)

    # attālums no punkta (x0;y0) līdz (x1;y1)
    if math.sqrt((x0 - x1)**2 + (y0 - y1)**2) > 5: # ja attālums starp šiem punktiem paliek mazāks nekā 5 (kvadrāta mala paliek mazāka nekā 5), tad rekursija pārtraucās
        kanva.create_line(x0, y0, x2, y2)
        kanva.create_line(x2, y2, x1, y1)
        kanva.create_line(x1, y1, x3, y3)
        kanva.create_line(x3, y3, x0, y0)

        zimet_rekursivi((x0 + x2) / 2, (y0 + y2) / 2, (x2 + x1) / 2, (y1 + y2) / 2, (x1 + x3) / 2, (y1 + y3) / 2, (x3 + x0) / 2, (y0 + y3) / 2)



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



logs = tkinter.Tk()
kanva = tkinter.Canvas(logs, width=600, height=600)
kanva.pack()

x0 = 50
y0 = 50

x1 = 550
y1 = 550
```

```
x2 = x0

y2 = y1


x3 = x1

y3 = y0


zimet_rekursivi(x0, y0, x2, y2, x1, y1, x3, y3)


logs.mainloop()
```

Testa piemēri

1)