**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)

Text

Description automatically generated

2)

Text

Description automatically generated

3)

Text

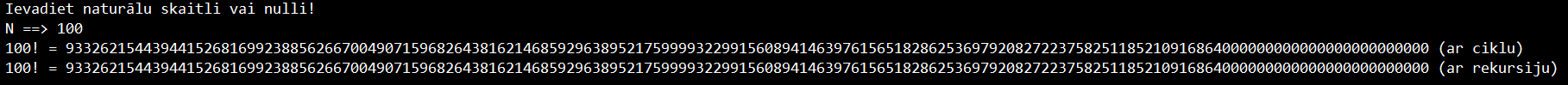
Description automatically generated

4)

Text

Description automatically generated

5)



6)

Text

Description automatically generated

**2. uzdevums**

Sastādīt programmu, kas aprēķina , ja zināms, ka

un

**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)

Text

Description automatically generated

2)

Text

Description automatically generated

3)

Text

Description automatically generated

4)

Text

Description automatically generated

5)

A screenshot of a computer

Description automatically generated with medium confidence

6)

Text

Description automatically generated

7)

A screenshot of a computer

Description automatically generated with medium confidence

8)

Text

Description automatically generated

9)

Text

Description automatically generated

10)

Graphical user interface, text, application, chat or text message

Description automatically generated

**3. uzdevums**

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

Venn diagram

Description automatically generated

**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)

Diagram, venn diagram

Description automatically generated

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

create\_circle(x, y, r)

draw\_circles(x + r, y, r // 2)

Diagram, venn diagram

Description automatically generated

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

create\_circle(x, y, r)

draw\_circles(x + r, y, r // 2)

Diagram, venn diagram

Description automatically generated

**4. uzdevums**

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

A picture containing fabric

Description automatically generated

**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…

Qr code

Description automatically generated

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

return…

A picture containing shape

Description automatically generated

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

return…

A picture containing shape

Description automatically generated

**5. uzdevums**

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

Diagram

Description automatically generated with low confidence

**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…

A picture containing bubble chart

Description automatically generated

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

return

Shape

Description automatically generated

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

Return…

A picture containing chart

Description automatically generated

**6. uzdevums**

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

A picture containing linedrawing, porcelain

Description automatically generated

**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**

Diagram, schematic

Description automatically generated

2) **rekursijas\_skaits == 15**

Diagram

Description automatically generated

3) **rekursijas\_skaits == 10**

Diagram

Description automatically generated

4) **rekursijas\_skaits == 5**

Diagram, schematic

Description automatically generated

5) **rekursijas\_skaits == 4**

Schematic

Description automatically generated

**PU1. uzdevums**

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

Diagram

Description automatically generated

**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**

Diagram

Description automatically generated

2) **rekursijas\_skaits == 6**

Diagram

Description automatically generated

3) **rekursijas\_skaits == 4**

Diagram, engineering drawing

Description automatically generated

4) **rekursijas\_skaits == 3**

A picture containing object, antenna

Description automatically generated

**PU2. uzdevums**

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

Circle

Description automatically generated

**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**

Diagram

Description automatically generated

1. **rekursijas\_skaits == 2**

Diagram

Description automatically generated

1. **rekursijas\_skaits == 1**

Diagram

Description automatically generated

1. **rekursijas\_skaits == 5**

A picture containing circle

Description automatically generated

1. **rekursijas\_skaits == 6**

A picture containing diagram

Description automatically generated