**1MPR01\_Simona\_Bļinova sb24037**

Gandrīz visos uzdevumos lietoju moduli ar funkcijām. (programmu mapē *funkcijas.py*)

**1.uzdevums**

Programma, kas nodrukā uz ekrāna formatētu pēc piemēra Paskāla trijstūri ar lietotāja noteiktu rindu skaitu.

**Kods:**

Funkcijas no *funckijas.py*:

# Funkcija faktoriāla aprēķinam

def faktorials(a):

faktoriala\_vertiba = 1

for i in range(1, a+1):

faktoriala\_vertiba \*= i

return faktoriala\_vertiba

# Funkcijas kombinācijas vērtības aprēķinam

def kombinacija(n, m):

if n == 0 and m == 0:

kombinacijas\_vertiba = 1

else:

n\_faktorials = faktorials(n)

m\_faktorials = faktorials(m)

n\_m\_starpiba = n - m

n\_m\_faktorials = faktorials(n\_m\_starpiba)

kombinacijas\_vertiba = n\_faktorials / (m\_faktorials \* n\_m\_faktorials)

return kombinacijas\_vertiba

# Funkcija pārbaudei, lai skaitlis būtu naturāls skaitlis

def naturals\_skaitlis(a):

# meģinājumu skaita skaitītājs (3 meģinājumi ierakstam)

meginajumi = 1

while meginajumi <= 3:

try:

a = int(a)

if a > 0:

return int(a)

except:

if meginajumi < 3:

meginajumi += 1

a = input('Ievadiet naturālo skaitli --> ')

else:

print('Programma beidz darbību!')

exit()

*1MPR01\_1\_Simona\_Blinova.py*:

import funkcijas

'''Ievades un ievades pārbaudes bloks'''

rindu\_skaits = input('Ievadiet Paskāla trijstūra rindu skaitu --> ')

rindu\_skaits = funkcijas.naturals\_skaitlis(rindu\_skaits)

'''Bloks atstarpju skaita aprēķinam'''

# Atstarpes starp skaitliem

lielaka\_skaitla\_kartas\_numurs = rindu\_skaits // 2

lielakais\_skaitlis = int(funkcijas.kombinacija(rindu\_skaits, lielaka\_skaitla\_kartas\_numurs))

lielaka\_skaitla\_garums = len(str(lielakais\_skaitlis))

# Atstarpes pirms pirma rindas elementa

pedeja\_rinda = ''

for kartas\_numurs in range(rindu\_skaits): # kartas numuru skaits ir vienāds ar rindas kārtas numuru

kombinacijas\_vertiba = int(funkcijas.kombinacija(rindu\_skaits-1, kartas\_numurs))

pedeja\_rinda = pedeja\_rinda + str(kombinacijas\_vertiba)

if rindu\_skaits-1 != kartas\_numurs:

pedeja\_rinda = pedeja\_rinda + ' ' \* lielaka\_skaitla\_garums

pedejas\_rindas\_garums = len(pedeja\_rinda)

'''Bloks Paskāla trijstūra izvadei'''

rinda = ''

for rindas\_numurs in range(rindu\_skaits-1):

for kartas\_numurs in range(rindas\_numurs+1):

kombinacijas\_vertiba = int(funkcijas.kombinacija(rindas\_numurs, kartas\_numurs))

rinda = rinda + str(kombinacijas\_vertiba)

if rindas\_numurs != kartas\_numurs:

rinda = rinda + ' ' \* lielaka\_skaitla\_garums

rindas\_garums = len(rinda)

atstarpju\_skaits = pedejas\_rindas\_garums - rindas\_garums

sakuma\_atsarpju\_skaits = atstarpju\_skaits // 2

rinda = ' ' \* sakuma\_atsarpju\_skaits + rinda

print(rinda)

rinda = ''

print(pedeja\_rinda)

Testa piemērs(1)

A group of numbers in a triangle shape

AI-generated content may be incorrect.

Testa piemērs(2)

A black text on a white background

AI-generated content may be incorrect.

Testa piemērs(3)

A triangle of numbers and letters

AI-generated content may be incorrect.

**2.uzdevums**

Programma, kas atrod vienādojuma ax3+by2+cz+d=0 visus veselos atrisinājumus intervālā [-10, 10], kur koeficientus a, b, c un d, kas arī veseli skaitli, nosaka lietotājs.

**Kods:**

Funkcijas no *funckijas.py*:

# Funkcija pārbaudei, lai skaitlis būtu vesels skaitlis

def vesels\_skaitlis(a):

# meģinajumu skaitītājs (3 meģinājumi pareizi ievadīt)

meginajumi = 1

while meginajumi <= 3:

try:

a = int(a)

return int(a)

except:

if meginajumi < 3:

meginajumi += 1

a = input('Ievadiet veselo skaitli vēlreiz --> ')

else:

print('Programma beidz darbību!')

exit()

*1MPR01\_2\_Simona\_Blinova.py*:

import funkcijas

'''Koeficientu ievades un pārbaudes bloks'''

koeficients\_a = input('Ievadiet veselo koeficientu a --> ')

koeficients\_a = funkcijas.vesels\_skaitlis(koeficients\_a)

koeficients\_b = input('Ievadiet veselo koeficientu b --> ')

koeficients\_b = funkcijas.vesels\_skaitlis(koeficients\_b)

koeficients\_c = input('Ievadiet veselo koeficientu c --> ')

koeficients\_c = funkcijas.vesels\_skaitlis(koeficients\_c)

koeficients\_d = input('Ievadiet veselo koeficientu d --> ')

koeficients\_d = funkcijas.vesels\_skaitlis(koeficients\_d)

'''Cikls atrisinājumu mēklēšanai'''

for x in range(-10, 11):

for y in range(11):

cz = koeficients\_a\*x\*x\*x + koeficients\_b\*y\*y + koeficients\_d

if koeficients\_c != 0:

z = int(round(cz/koeficients\_c))

else:

z = 0

if cz + z\*koeficients\_c == 0:

print(f'({x}, {y}, {z})')

Testa piemērs(1)

A black text on a white background

AI-generated content may be incorrect.

Testa piemērs(2)

A computer screen shot of words

AI-generated content may be incorrect.

Testa piemērs(3)

A white background with black numbers

AI-generated content may be incorrect.

**3.uzdevums**

Programma, kas aprēķina arcsin(x) izteiksmes vērtību pēc noradīta piemēra ar precizitāti 10-6, ko nosaka pēdējais saskaitāmais, ja x nosaka lietotājs un |x| < 1.

**Kods:**

*1MPR01\_3\_Simona\_Blinova.py*:

'''Funkciju bloks'''

# Funkcija, lai pārbaudītu ievades korektību

def vertibas\_intervals(a):

# meģinājumu skaita skaitītājs (3 meģinājumi ierakstam)

meginajumi = 1

while meginajumi <= 3:

try:

a = float(a)

if a < 1 and a > -1:

return float(a)

else:

raise Exception

except:

if meginajumi < 3:

meginajumi += 1

a = input('Ievadiet x vērtību (|x| < 1) vēlreiz --> ')

else:

print('Programma beidz darbību!')

exit()

'''Lietotāja ievades un datu pārabaudes bloks'''

x = input('Ievadiet x vērtību (|x| < 1) --> ')

x = vertibas\_intervals(x)

'''Izteiksmes aprēķina un precizitātes pārbaudes bloks'''

precizitate = 1e-6

saskaitamais = x

summa = saskaitamais

# Katra sskaitama sākuma sastāvdaļas

x\_dala\_saucejs = 1

x\_dala\_skaititajs = x

koeficientu\_dala\_saucejs = 1

koeficientu\_dala\_skaititajs = 1

pakape = 3

while abs(saskaitamais) >= precizitate:

# Cikls, kas noskaidro un piereizina skaitļus skaititājam vai saucējam, pirms daļas ar x vērtību kādā pakāpē

for skaitlis in range(1, pakape):

if skaitlis % 2 == 0:

koeficientu\_dala\_saucejs \*= skaitlis

else:

koeficientu\_dala\_skaititajs \*= skaitlis

x\_dala\_saucejs = pakape

x\_dala\_skaititajs = x\_dala\_skaititajs \* x \* x

saskaitamais = (koeficientu\_dala\_skaititajs/koeficientu\_dala\_saucejs) \* (x\_dala\_skaititajs/x\_dala\_saucejs)

summa += saskaitamais

pakape += 2

print(f'arcsin({x})={summa}')

Testa piemērs(1)



Testa piemērs(2)



Testa piemērs(3)

A number and symbols on a white background

AI-generated content may be incorrect.

**4.uzdevums**

Programma, kas nodrukā visus laimīgos pasta zīmogus (DD.MM.GGGG) kopš Kristus dzimšanas.

**Kods:**

Funkcijas no *funckijas.py*:

# Funkcija nuļļu pievienošanai

def nulles(a, p):

datuma\_vertiba = str(a)

vertibas\_garums = len(datuma\_vertiba)

if p == 'g':

atlikums = 4 - vertibas\_garums

elif p == 'm' or p == 'd':

atlikums = 2 - vertibas\_garums

datuma\_vertiba = '0' \* atlikums + datuma\_vertiba

return datuma\_vertiba

*1MPR01\_4\_Simona\_Blinova.py*:

import funkcijas

'''Cikls datumu pārbaudei un izvadei'''

for gads in range(1, 10000):

datums\_gads = funkcijas.nulles(gads, 'g')

# print(datums\_gads)

for menesis in range(1, 13):

datums\_menesis = funkcijas.nulles(menesis, 'm')

# print(datums\_menesis)

# match case, kas nosaka dienu skaitu menesī

match menesis:

case 1 | 3 | 5 | 7 | 8 | 10| 12:

dienas = 31

case 4 | 6 | 9 | 11:

dienas = 30

case 2:

if gads % 4000:

dienas = 28

elif gads % 400:

dienas = 29

elif gads % 100:

dienas = 28

elif gads % 4:

dienas = 29

else:

dienas = 28

for diena in range(1, dienas+1):

datums\_diena = funkcijas.nulles(diena, 'd')

# Tiek ierakstīta simbolu virkne ar apgrieztu meneša un dienas vērtību

parbaudes\_dala = datums\_menesis[::-1] + datums\_diena[::-1]

# print(parbaudes\_dala)

if parbaudes\_dala == datums\_gads:

print(f'{datums\_diena}.{datums\_menesis}.{datums\_gads}')

Testa piemēri šeit ir izgrieztas daļas no vienas izvades.

Testa piemērs(1)

A number with black numbers

AI-generated content may be incorrect.

Testa piemērs(2)

A number of digits on a white background

AI-generated content may be incorrect.

Testa piemērs(3)

A number with black numbers

AI-generated content may be incorrect.

**5.uzdevums**

Programma, kas atrod un izvada Mersena skaitļus, kas nav lielāki par 2\*1028.

**Kods:**

Funkcijas no *funckijas.py*:

# Funkcija pārbaudei vai skaitlis ir pirmskaitlis

def vai\_pirmskaitlis(a):

if a < 2:

return False

for i in range(2, a//2):

if a % i == 0:

return False

else:

return True

*1MPR01\_5\_Simona\_Blinova.py*:

import funkcijas

'''Skaitļu aprēķina un pārbaudes bloks'''

mazakais\_pirmskaitlis = 2

while mazakais\_pirmskaitlis < 2\*pow(10, 28):

if funkcijas.vai\_pirmskaitlis(mazakais\_pirmskaitlis) == True:

iespejamais\_mersena\_skaitlis = 2\*\*mazakais\_pirmskaitlis - 1

if funkcijas.vai\_pirmskaitlis(iespejamais\_mersena\_skaitlis) == True:

print(iespejamais\_mersena\_skaitlis)

mazakais\_pirmskaitlis += 1

Testa piemērs(1)

A number on a white background

AI-generated content may be incorrect.

Šeit testa piemērs ir tikai viens, jo lielākie skaitli tālāk tiek meklēti ļoti ilgi un visātrāk ir atrasti pirmie 7.