**10. praktiskais darbs. 2. semestris**

**1. uzdevums**

Sastādīt programmu, kas nodrošina determinanta ievadi, aprēķina tā vērtību un izvada uz ekrāna aprēķināto determinanta vērtību.

**Kods:**

# Programmas nosaukums: Determinants

# 1. uzdevums (1MPR10\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādit programmu, kas nodrošina determinanta ievadi, aprēķina tā vērtību un izvada uz ekrāna aprēķināto determinanta vērtību.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

import numpy

def determinants(x):

# Rekursīvi atrod matricas x determinanta vērtību.

# Atgriež determinanta vērtību kā skaitļi int no matricas x.

# x - kvadrātiska n x n divdimensijas numpy masīvs (matrica ar izmēriem n x n)

n = len(x)

if n == 1:

return x[0, 0]

det = 0

zime = 1

for i in range(n):

xx = numpy.empty((n - 1, n - 1))

for j in range(1, n):

z = 0

for k in range(n):

if k != i:

xx[j - 1, z] = x[j, k]

z = z + 1

y = determinants(xx)

det = det + zime \* x[0, i] \* y

zime = -zime # + - + - + ...

return det

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 ievade\_matrica\_float(n, m):

# Lietotājs var ievādīt nXm matricas elementus un funkcija atgriež divdimensijas masīvu ar n rindam un m kolonnam ar ievadītām vērtībam.

# Ievādītas vērtības ir reālas vērtības (matricas elementi varētu būt float).

# Glīti izvada atkarīgi no tas, cik ir nepieciešams starpes likt.

# n - naturāls skaitlis, kurš nosaka matricas rindas skaitu.

# m - naturāls skaitlis, kurš nosaka matricas kolonnas skaitu.

a = numpy.empty((n, m), dtype=float)

for i in range(n):

for j in range(m):

temp = input("Ievadiet matricas elememtu A(" + str(i) + "," + str(j) + ") ===> ")

while is\_real\_check(temp) == False:

temp = input("Kļūda! Ievadītais elements nav skaitlis!\nIevadiet matricas elememtu A(" + str(i) + "," + str(j) + ") ===> ")

a[i, j] = float(temp)

return a

def is\_real\_check(n):

# Pārbauda vai simbolu virkne ir reāls skaitlis vai nav.

# Atgriež True, ja tas ir reāls skaitlis (float).

# Atgriež False, ja tas nav reāls skaitlis (float).

# n - pārbaudāma simbolu virkne.

try:

float(n)

except:

return False

else:

return True

def matrix\_to\_string\_float(matrix):

# Atgriež matricas virknes attēlojumu, kur katra rinda ir atdalīta ar \n.

# Ja vērtība ir vesels skaitlis, tā tiek attēlota bez komata. Pretējā gadījumā tas tiek attēlots ar komatu.

# Funkcija atrod arī maksimālo vērtību garumu matricā un papildina nepieciešamus skaitļus ar tujšumiem " ", lai tie būtu pareizi izlīdzināti.

# matrix - matrica (divdimensijas numpy masīvs ar izmēriem n x m).

rindas = len(matrix)

kolonnas = len(matrix[0])

max\_len = 0

for i in range(rindas): # atrod max\_len, lai uzzinātu cik atkāpes ir nepieciešāmas

for j in range(kolonnas):

value = matrix[i][j]

if value == int(value):

value\_len = len(str(int(value)))

else:

value\_len = len(str(float(value)))

if value\_len > max\_len:

max\_len = value\_len

sv = ""

# izveido matricas virknes attēlojumu, kur katra rinda ir atdalīta ar \n un izlidzina to pēc skaitļa ar maksimālu garumu

for i in range(rindas):

for j in range(kolonnas):

value = matrix[i][j]

if value == int(value):

value = int(value)

else:

value = str(float(value))

atkape = " " \* (max\_len - len(str(value)))

sv += atkape + str(value)

if j < kolonnas - 1:

sv = sv + " "

sv = sv + "\n"

return sv

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

# Galvenā programmas daļa

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

n = input("Ievadiet kvadrātiskas matricas izmēru, kurai gribat atrast determinantu ==> ")

while not is\_natural(n):

n = input("Kļūda! Ievadiet determinanta izmēru ==> ")

n = int(n)

a = ievade\_matrica\_float(n, n)

print("\nJūsu ievadīta matrica A:")

print(matrix\_to\_string\_float(a))

print("det(A) =", determinants(a))

**Testa piemēri:**

1)

A picture containing text

Description automatically generated

2)

A screenshot of a computer

Description automatically generated with medium confidence

3)

Text

Description automatically generated

4)

Text

Description automatically generated

5)

Text

Description automatically generated with medium confidence

6)

Text

Description automatically generated

7)

A screenshot of a computer screen

Description automatically generated with medium confidence

8)

Text

Description automatically generated

9)

Text

Description automatically generated

**2. uzdevums**

Sastādīt programmu, kas nodrošina matricas ievadi, atrod inverso matricu, izmantojot adjunktu metodi, un izvada uz ekrāna gan doto, gan atrast inverso matricu.

Zināšanai: , kur ir matricas A adjunktu matrica.

**Kods:**

# Programmas nosaukums:

# 2. uzdevums (1MPR10\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas nodrošina matricas ievadi, atrod inverso matricu, izmantojot adjunktu metodi un izvada uz ekrāna gan doto,

# gan atrasto inverso matricu.

# Zināšanai: A^-1 = 1/detA \* ( A')^T, kur A' ir matricas adjunktu matrica.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

import numpy

def determinants(x):

# Rekursīvi atrod matricas x determinanta vērtību.

# Atgriež determinanta vērtību kā skaitļi no matricas x.

# x - kvadrātiska n x n divdimensijas numpy masīvs (matrica ar izmēriem n x n)

n = len(x)

if n == 1:

return x[0, 0]

det = 0

zime = 1

for i in range(n):

xx = numpy.empty((n - 1, n - 1))

for j in range(1, n):

z = 0

for k in range(n):

if k != i:

xx[j - 1, z] = x[j, k]

z = z + 1

y = determinants(xx)

det = det + zime \* x[0, i] \* y

zime = -zime # + - + - + ...

return det

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 ievade\_matrica\_float(n, m):

# Lietotājs var ievādīt nXm matricas elementus un funkcija atgriež divdimensijas masīvu ar n rindam un m kolonnam ar ievadītām vērtībam.

# Ievādītas vērtības ir reālas vērtības (matricas elementi varētu būt float)

# Glīti izvada atkarīgi no tas, cik ir nepieciešams starpes likt.

# n - naturāls skaitlis, kurš nosaka matricas rindas skaitu

# m - naturāls skaitlis, kurš nosaka matricas kolonnas skaitu

a = numpy.empty((n, m), dtype=float)

for i in range(n):

for j in range(m):

temp = input("Ievadiet matricas elememtu A(" + str(i) + "," + str(j) + ") ===> ")

while is\_real\_check(temp) == False:

temp = input("Kļūda! Ievadītais elements nav skaitlis!\nIevadiet matricas elememtu A(" + str(i) + "," + str(j) + ") ===> ")

a[i, j] = float(temp)

return a

def is\_real\_check(n):

# Pārbauda vai simbolu virkne ir reāls skaitlis vai nav.

# Atgriež True, ja tas ir reāls skaitlis (float).

# Atgriež False, ja tas nav reāls skaitlis (float).

# n - pārbaudāma simbolu virkne.

try:

float(n)

except:

return False

else:

return True

def matrix\_to\_string\_float(matrix):

# Atgriež matricas virknes attēlojumu, kur katra rinda ir atdalīta ar \n.

# Ja vērtība ir vesels skaitlis, tā tiek attēlota bez komata. Pretējā gadījumā tas tiek attēlots ar komatu.

# Funkcija atrod arī maksimālo vērtību garumu matricā un papildina nepieciešamus skaitļus ar tujšumiem " ", lai tie būtu pareizi izlīdzināti.

# matrix - matrica (divdimensijas numpy masīvs ar izmēriem n x m).

rindas = len(matrix)

kolonnas = len(matrix[0])

max\_len = 0

for i in range(rindas): # atrod max\_len, lai uzzinātu cik atkāpes ir nepieciešāmas

for j in range(kolonnas):

value = matrix[i][j]

if value == int(value):

value\_len = len(str(int(value)))

else:

value\_len = len(str(float(value)))

if value\_len > max\_len:

max\_len = value\_len

sv = ""

# izveido matricas virknes attēlojumu, kur katra rinda ir atdalīta ar \n un izlidzina to pēc skaitļa ar maksimālu garumu

for i in range(rindas):

for j in range(kolonnas):

value = matrix[i][j]

if value == int(value):

value = int(value)

else:

value = str(float(value))

atkape = " " \* (max\_len - len(str(value)))

sv += atkape + str(value)

if j < kolonnas - 1:

sv = sv + " "

sv = sv + "\n"

return sv

def transponenta(a):

# Atgriež transponētu matricu a b[i, j] = a[j, i]

# a - divdimensijas masīvs (matrica)

n = a.shape[0] # x axis

m = a.shape[1] # y axis

b = numpy.empty((m, n))

for i in range(m):

for j in range(n):

b[i, j] = a[j, i]

return b

def adjugate\_matrix(a):

# Atgriež dotās kvadrātmatricas a adjunktu matricu a\*.

# a - nXn divdimensijas masīvs (kvadrātmatrica)

n = a.shape[0]

if n == 1:

# Ja matrica is 1x1, tas adjunktu matrica ir tas paša matrica

return a

adj = numpy.zeros((n, n)) # Izveidojam tukšo mainīgu ar nullem, kura būs adjunktu matrica a\* matricai a

for i in range(n):

for j in range(n):

# Izveidojam paligmatricu, noņemot rinda i un kolonnu j no sākotnējās matricas a

paligmatrica = []

for k in range(n):

if k != i:

rinda = []

for l in range(n):

if l != j:

rinda.append(a[k, l])

paligmatrica.append(rinda)

# Konvertējam paligmatricu par NumPy masīvu

paligmatrica = numpy.array(paligmatrica)

# Aprēķinam paligmatricas determinantu

det = determinants(paligmatrica)

# Reizinām determinantu ar (-1)^(i+j)

zime = (-1) \*\* (i + j)

# Adjunktas matricas elements (i,j) tas ir determinants no palīgmatricas ar zimes reizinājumu (-1 vai +1)

adj[i, j] = det \* zime

return adj

def inverse\_matrix\_by\_def(a):

# Atgriež inverso matrīcu (divdimensijas masīvu) izmantojot transponenta, determinants un adjugate\_matrix funckijas

# a - divdimensijas masīvs (kvadrātiska matrica)

adj = adjugate\_matrix(a)

transponent\_adj\_matrix = transponenta(adj)

det = determinants(a)

if det == 0:

return "Neēksiste"

else:

inverse\_det = 1 / determinants(a)

inverse\_a = inverse\_det \* transponent\_adj\_matrix

return inverse\_a

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

# Galvenā programmas daļa

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

n = input("Ievadiet kvadrātiskas matricas izmēru, kurai gribat atrast invērsu matricu ==> ")

while not is\_natural(n):

n = input("Kļūda! Ievadiet kvadrātiskas matricas izmēru, kurai gribat atrast invērsu matricu ==> ")

n = int(n)

a = ievade\_matrica\_float(n, n)

print("\nJūsu ievadīta matrica A:")

print(matrix\_to\_string\_float(a))

print("Atrāsta invērsa matrica A^-1:")

inversa = inverse\_matrix\_by\_def(a)

if str(inversa) == "Neēksiste":

print("Matricai A neēksistē invērsa matrica A^-1.")

else:

print(matrix\_to\_string\_float(inversa))

**Testa piemēri:**

1)

Text

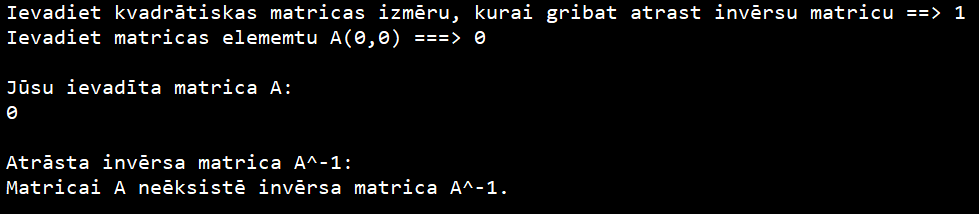
Description automatically generated

2)

A screenshot of a computer screen

Description automatically generated with medium confidence

3)



4)

A picture containing text

Description automatically generated

5)

Text

Description automatically generated

6)

Text

Description automatically generated

**3. uzdevums**

Sastādīt programmu, kas realizē Latloto 5 no 35 izlozi, nodrošina lietotāja vienā loterijas biļetē norādīto piecu skaitļu ievadi un paziņo par laimestu:

* Ja uzminēti 5 skaitļi, tad paziņo “Lielais laimests”
* Ja uzminēti 4 skaitļi, tad paziņo “Vidējais laimests”
* Ja uzminēti 3 skaitļi, tad paziņo “Mazais laimests”
* Ja citādi, tad paziņo “Nav laimesta”

**Kods:**

# Programmas nosaukums: Latloto

# 3. uzdevums (1MPR10\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas realizē Latloto 5 no 35 izlozi, nodrošina lietot vienā loterijas

# biļetē norādīto piecu skaitli ievadi un paziņo par laimestu:

# Ja uzminēti 5 skaitli, tad paziņo "Lielais laimests"

# Ja uzminēti 4 skaitļi, tad paziņo "Vidējais laimests"

# Ja uzminēti 3 skaitļi, tad paziņo "Mazais laimests"

# Ja citādi, tad paziņo "Nav laimesta".

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

import random

def random\_set\_numbers\_1\_to\_35():

# Atgriež kopu ar nejaušiem skaitļiem kuri neatkartojas no 1 līdz 35. [1, 35].

# a - set (kopa) - tukša kopa.

a = set()

while len(a) < 5:

b = random.randint(1, 35)

if b not in a:

a.add(b)

return a

def ievadiet\_n\_skaitlus\_seta(n, max\_num):

# Procedura kura ļauj ievādit n skaitļus un ieliekt tos kop (set'ā)

# n - cik skaitļus ievādīt lietotājam (int)

# max\_num - maksimālais skaitlis līdz kuram tiek veikta loterija (int)

a = set() # a - tukša kopa (set)

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

while True:

b = input("Ievadiet " + str(i) + ". skaitli ==> ")

try:

b = int(b)

if b < 1 or b > max\_num:

print("Skaitlim jābūt veselam skaitlim no 1 līdz " + str(max\_num) + ".")

elif b in a:

print("Šis skaitlis jau eksistē, lūdzu ievadiet citu skaitli!")

else:

a.add(b)

break

except ValueError:

print("Ievadei jābūt skaitlim!")

continue

return a

def laimests\_5\_35\_izloze(a, b):

# a - set (kopa) - nejauši izveidota kopa (jāizveido ar funkciju random\_set\_numbers\_1\_to\_35(a)).

# b - set (kopa) - cilvēka izvēlētie skaitļi kopā.

x = a.intersection(b)

len1 = len(x)

if len1 == 5:

return "Lielais laimests"

elif len1 == 4:

return "Vidējais laimests"

elif len1 == 3:

return "Mazais laimests"

else:

return "Nav laimesta"

def cik\_atminejat(a, b):

# Atgriež cik skaitļus Jūs laimējat (atrod kopas šķēlumu)

# a - set (kopa) - nejauši izveidota kopa (jāizveido ar funkciju random\_set\_numbers\_1\_to\_35(a)).

# b - set (kopa) - cilvēka izvēlētie skaitļi kopā.

x = a.intersection(b)

len1 = len(x)

return len1

def sort\_ievietosana\_augosa(a):

# Sakārto masīvu augoša secība un atgriež salīdzināšanas skaitu, lai sakārtotu masīvu

# Kārtošanas tiek izmantota ievietošanas metode (insertion sort)

# a - viendimensijas masīvs

n = len(a)

for i in range(1, n):

if a[i - 1] > a[i]:

x = a[i]

j = i

while a[j - 1] > x:

a[j] = a[j - 1]

j = j - 1

if j == 0:

break

a[j] = x

return a

def print\_set(a):

# Izvada (print) uz ekrāna kopu. Bet tas to neatgriež.

# Jāizsauc vienkarši print\_set(a)

# a - kopa (set)

elements = sort\_ievietosana\_augosa(list(a))

# Iterējam cauri elementu indeksiem un izdrukājiem tos, atdalot tos ar komatiem

for i in range(len(elements)):

if i == len(elements) - 1:

# Ja elements ir pēdējais, izdrukām to bez komata

print(elements[i])

else:

# Ja elements nav pēdējais, izdrukājam to ar komatu. end="", lai nebūtu pārejas uz jauno rindu kā \n

print(str(elements[i]) + ", ", end="")

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

# Galvenā programmas daļa

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

a = random\_set\_numbers\_1\_to\_35()

print("Sveicināti 'Latloto' 5 no 35 izloze!\n\nIevadiet savus skaitļus no 1 līdz 35.\nJā uzminēsiet 5 skaitļus, tad dabūsiet 'Lielo laimestu'\nJā uzminēsiet 4 skaitļus, tad dabūsiet 'Vidējo laimestu'\nJā uzminēsiet 3 skaitļus, tad dabūsiet 'Mazo laimestu'\nCitādi nav laimesta.\n")

#print("TESTĒŠANAI:", a) # TESTĒŠANAI (laimēto skaitļu set izvādīt)

b = ievadiet\_n\_skaitlus\_seta(5, 35)

print("\nJūsu izvēlētie skaitļi:")

print\_set(b)

print("\nIzlozētie skaitļi:")

print\_set(a)

atmineto\_skaits = cik\_atminejat(a, b)

if atmineto\_skaits == 1: # Pareizam locijumam

word = "skaitļi"

else:

word = "skaitļus"

print("\nJūs atminējat " + str(atmineto\_skaits) + " " + word + " no 5.")

print("\nJūsu laimests:")

print(laimests\_5\_35\_izloze(a, b))

**Testa piemēri:**

1)

Text

Description automatically generated

2)

Text

Description automatically generated

3)

Text

Description automatically generated

4)

Text

Description automatically generated

5)

Text

Description automatically generated

6)

Text

Description automatically generated

7)

Graphical user interface, text

Description automatically generated

**4. uzdevums**

Sastādīt programmu, kas realizē kāršu kavas izdali 4 spēlētājiem pa 6 kārtīm katram spēlētājam un paziņo, kādas kārtis katrs spēlētājs ir saņēmis.

**Kods:**

# Programmas nosaukums: Karšu kavas izdali

# 4. uzdevums (1MPR10\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas realizē kāršu kavas uzdali 4 spēlētājiem pa 6 kārtīm katram spēlētajam

# un paziņo, kādas kārtis katrs spēlētajs ir saņemis.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

# ♣ ♦ ♥ ♠

# 2 3 4 5 6 7 8 9 10 J Q K A

import random

def karsu\_izdale\_n\_speletajiem\_pa\_m\_kartim(n, m):

# Atgriež simbolu virknī šāda veidā ar spēlētaju kārtu un viņa kārtem:

# 1.spēlētājs: ♦6 ♦5 ♠J ♠5 ♠8 ♣A

# 2.spēlētājs: ♦8 ♠9 ♥6 ♠4 ♣9 ♥Q

# 3.spēlētājs: ♠7 ♣7 ♠6 ♦10 ♠K ♥4

# ...

# n.spēlētajs: ...

# To vajag print'ēt atsevišķi print(karsu\_izdale\_n\_speletajiem\_pa\_m\_kartim(4, 6))

# n - int - spēlētaju skaits (naturāls skaitlis)

# m - int - karšu skaits (naturāls skaitlis)

a = set(("♣", "♦", "♥", "♠"))

b = set(("2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A"))

c = set()

for i in a:

for j in b:

c.add(i + j)

res = ""

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

sv = str(i) + ".spēlētājs:"

for j in range(m):

sv = sv + " " + c.pop()

res = res + sv + "\n"

return res

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

# Galvenā programmas daļa

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

print("Kāršu kavas izdale 4 spēlētājiem pa 6 kārtīm katram spēlētājam:\n")

print(karsu\_izdale\_n\_speletajiem\_pa\_m\_kartim(4, 6))

**Testa piemēri:**

1)

A picture containing text, device, meter, control panel

Description automatically generated

2)

A picture containing text, screen, device, meter

Description automatically generated

3)

Text

Description automatically generated

4)

A picture containing text, meter, device, dark

Description automatically generated

5)

A picture containing text, device, meter, dark

Description automatically generated

6)

Text

Description automatically generated

**5. uzdevums**

Sastādīt programmu, kas realizē Keno loterijas izlozi un paziņo laimējošos skaitļus augošā secībā.

**Kods:**

# Programmas nosaukums: Keno loterijas izloze

# 5. uzdevums (1MPR10\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas realizē Keno loterijas izlozi un paziņo laimējošos skaitļus augošā secība.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

import random

def keno(n, m):

# Atgriež kopu ar n-tiem nejaušiem skaitļiem kuri neatkartojas no 1 līdz m. [1, m].

# a - set (kopa) - tukša kopa.

# Izvēlas n skaitļus no 1 līdz m, izmantojot sets. Tie neatkartosies

a = set()

while len(a) < n:

b = random.randint(1, m)

if b not in a:

a.add(b)

return a

def sort\_ievietosana\_augosa(a):

# Sakārto masīvu augoša secība un atgriež salīdzināšanas skaitu, lai sakārtotu masīvu

# Kārtošanas tiek izmantota ievietošanas metode (insertion sort)

# a - viendimensijas masīvs

n = len(a)

for i in range(1, n):

if a[i - 1] > a[i]:

x = a[i]

j = i

while a[j - 1] > x:

a[j] = a[j - 1]

j = j - 1

if j == 0:

break

a[j] = x

return a

def print\_set(a):

# Izvada (print) uz ekrāna kopu. Bet tas to neatgriež.

# Jāizsauc vienkarši print\_set(a)

# a - kopa (set)

elements = sort\_ievietosana\_augosa(list(a))

# Iterējam cauri elementu indeksiem un izdrukājiem tos, atdalot tos ar komatiem

for i in range(len(elements)):

if i == len(elements) - 1:

# Ja elements ir pēdējais, izdrukām to bez komata

print(elements[i])

else:

# Ja elements nav pēdējais, izdrukājam to ar komatu. end="", lai nebūtu pārejas uz jauno rindu kā \n

print(str(elements[i]) + ", ", end="")

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

# Galvenā programmas daļa

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

# Izvadam uz ekrāna laimējošos skaitļus

print("Keno loterijas izlozes laimesta skaitļi augošā secībā:")

keno = keno(20, 62)

print\_set(keno)

**Testa piemēri:**

1)

A picture containing graphical user interface

Description automatically generated

2)

Text

Description automatically generated with medium confidence

3)

Text

Description automatically generated

4)



5)

Text

Description automatically generated

6)

Text

Description automatically generated with medium confidence

**PU1. uzdevums**

Papildināt 5. Uzdevumā realizēto Keno loterijas izlozi ar laimestu sadali.

**Kods:**

# Programmas nosaukums: Keno loterijas izloze ar laimesto sadalījumu

# 5. uzdevums (1MPR10\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Papildināt 5.uzdevumā realizēto Keno loterijas izlozi ar laimestu sadali.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

# Avots: https://www.latloto.lv/lv/keno/

# Noteikumi: https://www.latloto.lv/lv/page/view/2695

import numpy

import random

def is\_natural\_and\_less\_than\_10(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 and int(n) < 11:

return True

else:

return False

def ievadiet\_n\_skaitlus\_seta(n, max\_num):

# Procedura kura ļauj ievādit n skaitļus un ieliekt tos kop (set'ā)

# Atgriež jau aizpildīto kopu (set'u) a ar lietotāja ievādītajiem naturāliem skaitliem robēžas no 1 līdz max\_num ieskaitot.

# n - cik skaitļus ievādīt lietotājam (int)

# max\_num - maksimālais skaitlis līdz kuram tiek veikta loterija (int)

a = set()

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

while True:

b = input("Ievadiet " + str(i) + ". skaitli ==> ")

try:

b = int(b)

if b < 1 or b > max\_num:

print("Skaitlim jābūt veselam skaitlim no 1 līdz " + str(max\_num) + ".")

elif b in a:

print("Šis skaitlis jau eksistē, lūdzu ievadiet citu skaitli!")

else:

a.add(b)

break

except ValueError:

print("Ievadei jābūt skaitlim!")

continue

return a

def speles\_likmes():

# Paprasam izvēlēties spēlēs likmi no piedāvātiem.

# Pārbauda to, vai tāda eksistē un atgriež to, kā float.

while True:

n = input("Izvēlēties, Jūsu spēles likmi € ==> ")

try:

float(n)

except:

print("Kļūda! Ievādiet reālo likmi no piedāvatiem!")

else:

n = float(n)

if n == 0.2:

likme = 0.2

return likme

elif n == 0.3:

likme = 0.3

return likme

elif n == 0.5:

likme = 0.5

return likme

elif n == 1:

likme = 1

return likme

elif n == 2:

likme = 2

return likme

elif n == 3:

likme = 3

return likme

elif n == 5:

likme = 5

return likme

elif n == 10:

likme = 10

return likme

else:

print("Kļūda! Ievādiet likmi no piedāvatiem!")

def laimestu\_matrica(izveleto\_skaits, atmineto\_skaits):

# Atgriež reizinātāju pamatojoties uz laimestu sadalījuma matricas.

# Reizinātajs ir atkarīgs no tā, cik skaitļus jus izvēlējaties un cik jus atminējat.

# Reizinātāji ir pārādīta šājā matricā.

laimesti = numpy.array([[0, 0, 0, 1, 1, 1, 1, 1, 1, 1],

[1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0],

[0, 4.5, 1, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 8, 1, 1, 0, 0, 0, 0, 0],

[0, 0, 0, 20, 2, 2, 1, 0, 0, 0],

[0, 0, 0, 0, 45, 12, 3, 3, 1, 1],

[0, 0, 0, 0, 0, 175, 30, 5, 2, 2],

[0, 0, 0, 0, 0, 0, 700, 100, 40, 5],

[0, 0, 0, 0, 0, 0, 0, 3000, 350, 55],

[0, 0, 0, 0, 0, 0, 0, 0, 10000, 550],

[0, 0, 0, 0, 0, 0, 0, 0, 0, 60000]])

reizinatajs = laimesti[atmineto\_skaits, izveleto\_skaits - 1] # Izvēlāmies nepieciešāmo rūtiņu un tas arī būs reizinātajs.

return reizinatajs

def naudas\_balva(likme, reizinatajs):

# Atgriež naudas balvu likmi reizinot ar reizinātaju.

# likme - likme (float reāls skaitlis lielāks par 0, piemēram,

# 0.20

# 0.30

# 0.50

# 1.00

# 2.00

# 3.00

# 5.00

# 10.00)

# reizinātajs - float reāls skaitlis lielāks vai vienāds par 0

return likme \* reizinatajs

def random\_set\_numbers\_skaita\_n\_from\_1\_to\_m(n, m):

# Atgriež kopu ar n nejaušiem skaitļiem, kuri neatkartojas no 1 līdz m. [1, m].

# n - cik nejaušus skaitļus vajag loterijai

# m - kāda ir augšēja robeža skaitlim, kuru vajadzīgi uzminēt (vislielākais iespējamais skaitlis loterija)

a = set()

while len(a) < n:

b = random.randint(1, m)

if b not in a:

a.add(b)

return a

def cik\_atminejat(a, b):

# Atgriež cik skaitļus Jūs laimējat (atrod kopas šķēlumu)

# a - set (kopa) - nejauši izveidota kopa (jāizveido ar funkciju random\_set\_numbers\_1\_to\_35(a)).

# b - set (kopa) - cilvēka izvēlētie skaitļi kopā.

x = a.intersection(b)

len1 = len(x)

return len1

def print\_set(a):

# Izvada (print) uz ekrāna kopu. Bet tas to neatgriež.

# Jāizsauc vienkarši print\_set(a)

# a - kopa (set)

elements = sort\_ievietosana\_augosa(list(a))

# Iterējam cauri elementu indeksiem un izdrukājiem tos, atdalot tos ar komatiem

for i in range(len(elements)):

if i == len(elements) - 1:

# Ja elements ir pēdējais, izdrukām to bez komata

print(elements[i])

else:

# Ja elements nav pēdējais, izdrukājam to ar komatu. end="", lai nebūtu pārejas uz jauno rindu kā \n

print(str(elements[i]) + ", ", end="")

def sort\_ievietosana\_augosa(a):

# Sakārto masīvu augoša secība un atgriež salīdzināšanas skaitu, lai sakārtotu masīvu

# Kārtošanas tiek izmantota ievietošanas metode (insertion sort)

# a - viendimensijas masīvs

n = len(a)

for i in range(1, n):

if a[i - 1] > a[i]:

x = a[i]

j = i

while a[j - 1] > x:

a[j] = a[j - 1]

j = j - 1

if j == 0:

break

a[j] = x

return a

def izlozu\_skaits():

# Paprasam izvēlēties izložu skaitu no piedāvātiem.

# Pārbauda to, vai tāda eksistē un atgriež to, kā float.

while True:

n = input("Izvēlēties, izložu skaitu ==> ")

try:

int(n)

except:

print("Kļūda! Ievādiet reālo izložu skaitu no piedāvatiem!")

else:

n = int(n)

if n == 1:

izlozu\_skaits = 1

return izlozu\_skaits

elif n == 2:

izlozu\_skaits = 2

return izlozu\_skaits

elif n == 3:

izlozu\_skaits = 3

return izlozu\_skaits

elif n == 4:

izlozu\_skaits = 4

return izlozu\_skaits

elif n == 6:

izlozu\_skaits = 6

return izlozu\_skaits

elif n == 12:

izlozu\_skaits = 12

return izlozu\_skaits

elif n == 21:

izlozu\_skaits = 21

return izlozu\_skaits

else:

print("Kļūda! Ievādiet izložu skaitu no piedāvatiem!")

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

# Galvenā programmas daļa

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

n = input("Izvēlēties, cik skaitļus nosvītrot (no 1 līdz 10) ==> ")

while not is\_natural\_and\_less\_than\_10(n):

n = input("Kļūda! Izvēlēties, cik skaitļus nosvītrot (no 1 līdz 10) ==> ")

print("Ievādiet Jūsu skaitļus. Skaitlim jābūt veselam skaitlim no 1 līdz 62.")

n = int(n)

rezultats = random\_set\_numbers\_skaita\_n\_from\_1\_to\_m(20, 62)

# print("TESTĒŠANAI:") # TESTĒŠANAI

# print\_set(rezultats) # TESTĒŠANAI

players\_numbers = ievadiet\_n\_skaitlus\_seta(n, 62)

print("\nSpēles likmes:\n0.20 €\n0.30 €\n0.50 €\n1.00 €\n2.00 €\n3.00 €\n5.00 €\n10.00 €\n")

likme = speles\_likmes()

print("\nIzložu skaits:\n1\n2\n3\n4\n6\n12\n21\n")

piedalisanas\_reizes = izlozu\_skaits()

print("\nJūsu skaitļi:")

print\_set(players\_numbers)

total\_laimests = 0

for i in range(piedalisanas\_reizes):

rezultats = random\_set\_numbers\_skaita\_n\_from\_1\_to\_m(20, 62) # Noņemt testēšanai

print("\nIzlozētie skaitļi:")

print\_set(rezultats)

atmineto\_skaits = cik\_atminejat(players\_numbers, rezultats)

if atmineto\_skaits == 1: # Pareizam locijumam

word = " skaitļi."

else:

word = " skaitļus."

print("\nJūs atminējat " + str(atmineto\_skaits) + word)

reizinatajs = laimestu\_matrica(n, atmineto\_skaits)

print("Reizinātājs:", reizinatajs)

laimests = naudas\_balva(likme, reizinatajs)

print("Jūsu laimests:", laimests, "€")

total\_laimests += laimests

if piedalisanas\_reizes == 1:

pass

elif i + 1 == piedalisanas\_reizes:

if piedalisanas\_reizes == 21: # Pareizam locijumam

word = "izlozi"

else:

word = "izlozem"

print("\nJūsu kopējais laimests par", piedalisanas\_reizes, word, "ir:", total\_laimests, "€")

**Testa piemēri:**

1)

Text

Description automatically generated

2)

Text

Description automatically generated

3)

Text

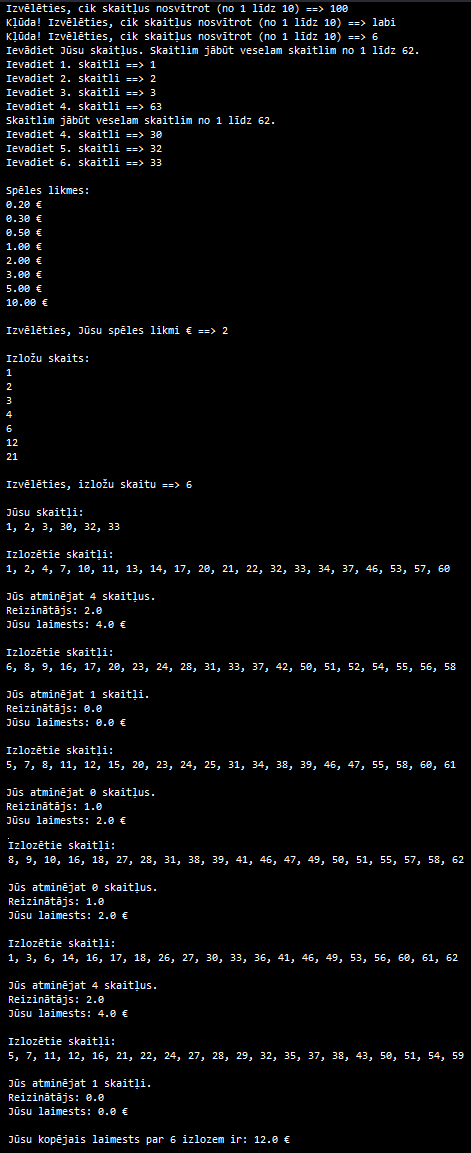
Description automatically generated

4)

Text

Description automatically generated

5)



6)

Text

Description automatically generated