**1.uzd**

def bank(summa):

summa\_banknoti = int(summa)

summa\_centi = round((summa - summa\_banknoti), 2)

# print(summa\_centi)

saraksts\_banknotu\_skaitu = [0, 0, 0, 0, 0, 0, 0]

banknotes = [500, 200, 100, 50, 20, 10, 5]

monetas = [2, 1]

saraksts\_monetu\_skaitu = [0, 0]

centi = [50, 20, 10, 5, 2, 1]

saraksts\_centu\_skaitu = [0, 0, 0, 0, 0, 0, 0]

for i in range(len(banknotes)):

if summa\_banknoti >= banknotes[i]:

saraksts\_banknotu\_skaitu[i] += 1

summa\_banknoti = summa\_banknoti - banknotes[i]

for i in range(len(monetas)):

if summa\_banknoti >= monetas[i]:

saraksts\_monetu\_skaitu[i] += 1

summa\_banknoti = summa\_banknoti - monetas[i]

summa\_centi = 100 \* summa\_centi

for i in range(len(centi)):

if summa\_centi >= centi[i]:

saraksts\_centu\_skaitu[i] += 1

summa\_centi = summa\_centi - centi[i]

for i in range(7):

if saraksts\_banknotu\_skaitu[i] == 0:

pass

else:

print(banknotes[i], "EUR banknošu skaits:", saraksts\_banknotu\_skaitu[i])

for i in range(2):

if saraksts\_monetu\_skaitu[i] == 0:

pass

else:

print(monetas[i], "EUR monetas skaits:", saraksts\_monetu\_skaitu[i])

for i in range(7):

if saraksts\_centu\_skaitu[i] == 0:

pass

else:

print(centi[i], "euro-centu monetas skaits:", saraksts\_centu\_skaitu[i])

eur\_summa = float(input("Ievadiet EUR summu ==> "))

bank(eur\_summa)

**2.uzd**

# rindu, kolonnu un diagonaļu summa

import numpy

def is\_whole(n):

# Pārbauda vai simbolu virkne ir vesels skaitlis vai nav

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

# n - simbolu virkne, kuru pārbauda.

try:

n = int(n)

except:

return False

else:

return True

def izvade\_matrica\_int(a):

# Atgriež divdimensiju masīvu (matricu), tabulas veidā, str formāta, kur katra rinda ir atdalīta ar jauno rindkopu

# a - divdimensijas masīvs (matrica)

n = a.shape[0] # "x axis" masīva izmēri

m = a.shape[1] # "y axis" masīva izmēri

s = ""

for i in range(n):

for j in range(m):

s = s + "{:8.0f}".format(int(a[i, j]))

s = s + "\n"

return s

def ievade\_matrica(n, m):

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

# n - naturāls skaitlis, kurš nosaka matricas rindas skaitu

# m - naturāls skaitlis, kurš nosaka matricas kolonnas skaitu

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

for i in range(n):

for j in range(m):

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

while is\_whole(temp) == False:

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

a[i, j] = int(temp)

return a

def matrix\_row\_sum(a):

# rindas summa

b = a.shape[0]

c = numpy.zeros(b)

for i in range(b):

summa = 0

for j in range(b):

summa = summa + a[i, j]

c[i] = summa

return c

def matrix\_col\_sum(a):

# kolonnu summa

b = a.shape[0]

c = numpy.zeros(n)

for i in range(b):

summa = 0

for j in range(b):

summa = summa + a[j, i]

c[i] = summa

return c

def matrix\_diag\_sum(a):

# diagonalu summa

b = a.shape[0]

c = numpy.zeros(n)

summa = 0

for i in range(b):

summa = summa + a[i, i]

c[0] = summa

summa = 0

for i in range(b):

summa = summa + a[i, n - 1 - i]

c[1] = summa

return c

def new\_matrix(a, c, e, d):

n = a.shape[0]

new\_a = numpy.zeros((n + 2, n + 2))

new\_a[1:n + 1, 1:n + 1] = a # centrs

new\_a[1:n + 1, 0] = d

new\_a[0, 1:n + 1] = e

new\_a[1:n + 1, -1] = d

new\_a[-1, 1:n + 1] = e

new\_a[n + 1, n + 1] = c[0]

new\_a[n + 1, 0] = c[1]

new\_a[0, n + 1] = c[1]

new\_a[0, 0] = c[0]

return new\_a

# galvenā programma

n = int(input("Ievadiet kvadrātitskas matricas izmēru ==> "))

a = ievade\_matrica(n, n)

b = izvade\_matrica\_int(a)

print("Ievadītā matrica:")

print(b)

# print(b)

# izsauc summu funkcijas

c = matrix\_diag\_sum(a)

d = matrix\_row\_sum(a)

e = matrix\_col\_sum(a)

# print(c)

# print(d)

# print(e)

new\_a = new\_matrix(a, c, e, d)

new\_b = izvade\_matrica\_int(new\_a)

print("Jauna matrica:")

print(new\_b)

**3.uzd**

import numpy

def is\_whole(n):

# Pārbauda vai simbolu virkne ir vesels skaitlis vai nav

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

# n - simbolu virkne, kuru pārbauda.

try:

n = int(n)

except:

return False

else:

return True

def ievade\_matrica\_sim(n, m):

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

# n - naturāls skaitlis, kurš nosaka matricas rindas skaitu

# m - naturāls skaitlis, kurš nosaka matricas kolonnas skaitu

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

for i in range(n):

for j in range(m):

if i == j or i > j:

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

while is\_whole(temp) == False:

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

a[i, j] = int(temp)

return a

def izvade\_matrica\_int(a):

# Atgriež divdimensiju masīvu (matricu), tabulas veidā, str formāta, kur katra rinda ir atdalīta ar jauno rindkopu

# a - divdimensijas masīvs (matrica)

n = a.shape[0] # "x axis" masīva izmēri

m = a.shape[1] # "y axis" masīva izmēri

s = ""

for i in range(n):

for j in range(m):

s = s + "{:8.0f}".format(int(a[i, j]))

s = s + "\n"

return s

def make\_symmetrical(a):

n = a.shape[0]

m = a.shape[1]

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

for i in range(m):

for j in range(n):

if i < j:

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

elif i > j or i == j:

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

return b

n = int(input("Ievadiet matricas izmēru ==> "))

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

print("Jānīša matrica pirms atjaunošanas")

print(izvade\_matrica\_int(a))

print("Jānīša matrica pēc atjaunošanas")

sim\_a = make\_symmetrical(a)

print(izvade\_matrica\_int(sim\_a))

**4.uzd**

import random

def izloze(cik, nocik): # cik == 5, nocik == 35

v = set()

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

v.add(i)

k = set()

for i in range(cik):

b = random.choice(list(v))

v.remove(b)

k.add(b)

return k

def again\_izloze(kopa):

# funkcija veic atkārtotu vienas lodītes lozēšanu un pievieno to izlozēto lodīšu kopai

nejausi = random.randint(1, 35)

for nejausi in kopa:

nejausi = random.randint(1, 35)

kopa.add(nejausi)

return kopa

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 izvade(a):

# funkcija nodrošina datu glītu izvadi simbolu virknes formā

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

sv = ""

for i in range(len(a)):

if i < len(a) - 1:

sv = sv + str(a[i]) + " "

else:

sv = sv + str(a[i])

return sv

def main\_izloze():

# galvenā programma

jan\_lodites = izloze(5, 35) # jānīša izlozētās lodītes

# print(jan\_lodites)

peter\_lodites = izloze(5, 35) # pēterīša izlozētās lodītes

laimests = 1024

tries = 1

kopigas\_lodites = jan\_lodites.intersection(peter\_lodites) # kopīgās lodītes

# ja šķēluma kopas garums ir 5, tātad izlozētas vienādas lodītes

if len(kopigas\_lodites) == 5:

pass # laimest = 1024

# tries = 1

else:

# ja šķēluma kopas garums nav pieci, tad lozē papildus lodītes

while len(kopigas\_lodites) < 5:

again\_izloze(jan\_lodites)

again\_izloze(peter\_lodites)

kopigas\_lodites = jan\_lodites.intersection(peter\_lodites)

else:

i = len(jan\_lodites) - 5 # aprēķina kāpinātāju laimesta aprēķināšanai

tries = i + tries

laimests = laimests / 2\*\*i

return (laimests, tries, jan\_lodites, peter\_lodites, kopigas\_lodites)

# rezultāta izvade

galv\_izloze = main\_izloze()

# print(galv\_izloze[1])

print(str(galv\_izloze[1]) + ". meginajumi tika izdarīti, kamēr netika izvilktas piecas kopīgas lodītes.")

#print("Tika izdarīti", tries, "mēģinājumi.")

print("Jānītis izvilka lodītes: " + str(izvade(galv\_izloze[2])))

print("Pēterītis izvilka lodītes: " + str(izvade(galv\_izloze[3])))

print("Kopīgās lodītes: " + str(izvade(galv\_izloze[4])))

formattets\_laimests = "{:.2f}".format(galv\_izloze[0])

print("Laimests: " + formattets\_laimests + " EUR")

**5.uzd**

Здесь алгоритм шифрования какой-то

#1 #2 #3 #4 #3 #4 #5 #5 #2 #4 #5 #3 #10 #6 #7 #7 #8 #5 #6 #8 #7 #8 #11 #10 #8 #8 #3 #11 #7

|  |  |
| --- | --- |
| **Nr** | **Frāze** |
| **1** | **A** |
| **2** | **B** |
| **3** | **AB** |
| **4** | **BA** |
| **5** | **ABB** |
| **6** | **BAA** |
| **7** | **ABBA** |
| **8** | **AA** |
| **9** | **ABBAB** |
| **10** | **BB** |
| **11** | **BBB** |
| **12** | **BAAB** |

**6.uzd**

# Open the files: datne1 for reading and datne2 for writing

datne1 = open("C:\\Users\\User\\Desktop\\OGAS.txt", mode="r", encoding="utf-8")

datne2 = open("C:\\Users\\User\\Desktop\\POGAS.txt", mode="w", encoding="utf-8")

simbols = datne1.read(1)

while simbols != "":

space\_count = 0

while simbols == " ":

space\_count += 1

simbols = datne1.read(1)

if space\_count > 0:

simbols = " " + simbols

datne2.write(simbols)

simbols = datne1.read(1)

datne1.close()

datne2.close()