**11. praktiskais darbs. 2. semestris**

**1. uzdevums**

Sastādīt programmu, kas simulē eksāmena norisi, pieņemot, ka grupā ir 54 studenti, kas apzīmēti ar skaitļiem 1, 2, 3, ... 53, 54, kuriem tiek piedāvātas 26 eksāmena biļetes, kas apzīmētas ar lielajiem latīņu alfabēta burtiem. Eksāmenu sāk 10 uz labu laimi izvelēti, studenti, kuri katrs saņem tiešu vienu no 26 piedāvātajām biļetēm. Kad 1. students atbildējis, viņa biļete tiek atlikta atpakaļ eksāmena biļešu "kaudzē" un nāk nākamais students un izvēlās uz labu laimi vienu eksāmena biļeti no eksāmena biļešu "kaudzes". Tā process tiek atkārtots, kamēr visi studenti ir noeksaminēti. Paziņot uz ekrāna, kādā secība studenti tika eksaminēti. Paziņot uz ekrāna, kādā secība studenti tika eksaminēti un kādu eksāmena biļeti katrs students saņēma.

NB! Uzdevuma atrisināšanai jāizmanto kopas.

**Kods:**

# Programmas nosaukums: Eksāmena norises simulācija

# 1. uzdevums (1MPR11\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas simulē eksāmena norisi, pieņemot, ka grupā ir 54 studenti, kas apzīmēti ar skaitļiem 1, 2, 3, ... 53, 54,

# kuriem tiek piedāvātas 26 eksāmena biļetes, kas apzīmētas ar lielajiem latīņu alfabēta burtiem.

# Eksāmenu sāk 10 uz labu laimi izvelēti, studenti, kuri katrs saņem tiešu vienu no 26 piedāvātajām biļetēm.

# Kad 1. students atbildējis, viņa biļete tiek atlikta atpakaļ eksāmena biļešu "kaudzē" un nāk nākamais students un izvēlās uz labu laimi vienu eksāmena biļeti no eksāmena biļešu "kaudzes".

# Tā process tiek atkārtots, kamēr visi studenti ir noeksaminēti. Paziņot uz ekrāna, kādā secība studenti tika eksaminēti un kādu eksāmena biļeti katrs students saņēma.

# NB! Uzdevuma atrisināšanai jāizmanto kopas.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

import random

def choose\_pair(students\_prepare):

# Atgriež pirmu masīva students\_prepare lementu un nodzes to no saraksta students\_prepare.

# students\_prepare - saraksts (list) ar studentiem telpā (kuri gatavojas eksāmenam un sež ar biļētem).

# students\_prepare ir saraksts ar kortēžiem iekšā. [(studenta numurs, biļete), (studenta numurs, biļete), ... (studenta numurs, biļete)],

first\_pair = students\_prepare[0] # Paņem pirmo kortēžu ar studentu un biļeti.

students\_prepare.remove(first\_pair) # Nodzēs studentu no saraksta kuri gatavojas (jo viņš jau atbild),

return first\_pair

def choose\_random\_pair(students\_prepare, students, exam\_tickets):

# Atgriež nejaušu (student, exam\_ticket) kortēžu, no saraksta students (studenti, kas vel nav nokārtojusi eksāmenu un vēl nav eksāmena telpā).

# Nejauši izvēlas studentu no tiem, kas vēl nav nokārtojusi eksāmenu un nesež telpā. Atgriež kortežu ar studentu un eksāmena biļeti. (student, exam\_ticket),

# students\_prepare - saraksts (list) ar studentiem telpā (kuri gatavojas eksāmenam un sež ar biļētem).

# students - kopa ar visiem studenti kuri nav eksāmena telpā un pagaidam nav nokārtojusi eksāmenu.

# exam\_tickets - kopa ar visam biļetem no A līdz Z.

if len(students) == 0:

return None

student = students.pop()

exam\_ticket = exam\_tickets.pop()

return (student, exam\_ticket)

def print\_pairs(pairs):

# Glīti izvāda sarakstu pairs, kas reprēzentē studentus ar biļētem kuri sēž telpā un gatavojas eksāmenam.

# Piemērs:

# Ievade tāds saraksts: [(36, S), (50, T), (37, D), (42, R), (19, W), (22, U), (41, M), (44, Q), (52, P), (38, H)].

# Funkcija atgriež tādu simbolu virkni: 36-S, 50-T, 37-D, 42-R, 19-W, 22-U, 41-M, 44-Q, 52-P, 38-H

# pairs - saraksts ar kortežiem ar divam vērtībam [(students, biļete), (students, biļete), ... , (students, biļete)].

for i in range(len(pairs)):

if i == len(pairs) - 1:

print(str(pairs[i][0]) + "-" + str(pairs[i][1]))

else:

print(str(pairs[i][0]) + "-" + str(pairs[i][1]) + ", ", end="")

def convert\_int\_to\_str\_in\_set\_in\_range(n, m, kopa):

# Metode (neko neatgriež), kura izmaina visas int vērtības kopā uz str.

# Piemēram:

# {5, 33, 14, 13, 11} -> {'5', '33', '14', '13', '11'}.

# convert\_int\_to\_str\_in\_set\_in\_range(1, 55, students).

# kopa - set, kopa, kurai gribāt visas int vērtības izmainīt uz str. Piemēram: 1 -> "1"

# n - int, no cik (parasti no 1).

# m - int, līdz cik līdz m-1 (šeit līdz 55, bet funkcijai izmainīs int uz str līdz 54).

for i in range(n, m):

kopa.add(str(i))

def exam\_tickets\_words\_in\_range(n, m, exam\_tickets):

# Metode (neko neatgriež), kura piepildina tukšo kopu exam\_tickets ar simboliem no ASCII koda no n līdz m str veidā.

# exam\_tickets - tukša kopa, kurai jābut kopai ar eksāmena biļētiem no A līdz Z.

# n - int, no cik (ASCII kods simbols).

# m - int, līdz cik (ASCII kods).

# (65, 91) - lielajiem latiņu burtiem.

# Piemērs: exam\_tickets\_words\_in\_range(65, 91, exam\_tickets).

for i in range(n, m):

exam\_tickets.add(chr(i))

def start\_students\_number(n, students, exam\_tickets):

# Atgriež students\_prepare sarakstu, kura sastav no kortēžiem [(student, exam\_ticket), (student, exam\_ticket), ... , (student, exam\_ticket)].

# un šeit ir n šādu kortežu (student, exam\_ticket).

# n - int, cik "start" studentus paņemt? (10 šajā gadījumā). Tas ir cik studentus "izlozēsim" kartot eksāmenu pirmājiem.

# students - set, kopa ar visiem studentiem, kuri nav ienākuši eksāmenas telpā un vēl nav nokārtojusi eksāmenu. No viņiem paņemsim pirmus 10 studentu.

# exam\_tickets - set, kopa ar visiem eksāmena biļētem, kuri nav pie studentiem.

for i in range(n):

student = students.pop()

exam\_ticket = exam\_tickets.pop()

students\_prepare.append((student, exam\_ticket))

return students\_prepare

def exam\_simulation(students\_prepare):

# Simulē eksāmena norisināšanas.

# students\_prepare - saraksts veidā [(student, exam\_ticket), (student, exam\_ticket), ... , (student, exam\_ticket)].

# Kad 1. students atbildējis, viņa biļete tiek atlikta atpakaļ eksāmena biļešu "kaudzē" un nāk nākamais students un izvēlās uz labu laimi vienu eksāmena biļeti no eksāmena biļešu "kaudzes".

# Tā process tiek atkārtots, kamēr visi studenti ir noeksaminēti.

# Paziņo uz ekrāna, kādā secība studenti tika eksaminēti un kādu eksāmena biļeti katrs students saņēma.

exam = True # Karogs ir True, kamēr students\_prepare != 0 (kāmer eksamenācijas telpā ir palikuši cilvēki, tad eksāmens turpinās).

while exam: # Kamēr eksāmens ir True

if len(students\_prepare) == 0: # Ja nav nevienu cilvēku eksamenācijas telpā, tad eksāmens beidzās.

print("Eksāmens beidzās!")

exam = False # eksāmens beidzās.

else:

pair = choose\_pair(students\_prepare) # Izvēlas pirmo pāriti (kreiso parīti) (students, biļete) no studentiem, kuri gatavojas eksāmenam un sež telpā ar biļetem.

print("Atbild " + str(pair[0]) + ". students ar biļeti " + str(pair[1]) + ".") # Izvadam informāciju, kurš students atbild (kreisāis) un ar kādu biļeti viņš ir.

exam\_tickets.add(pair[1]) # Pievienojam biļeti no korteža eksāmena biļetes kaudzītei (studenta biļeti, kurš ir atbildējis uz jautājumu).

print("Biļete", pair[1], "tiek atlikta biļešu \"kaudzē\".\n") # Izvadam informāciju, kura biļete tika atlikta kaudzē.

if len(students) == 0: # Ja visi studenti jau ir ienākuši eksāmenā telpā, tad jau mēs vairs nevienu neaicināsim atkāl uz eksāmenu. (jo tie ir pēdējie studenti, kuri karto eksāmenu).

print\_pairs(students\_prepare) # Glīti izvadīt informāciju lietotājam par to, kuri cilvēki sēž un gatavojas eksāmenam iekšā telpā.

pair = choose\_pair(students\_prepare) # Izvēlas pirmo pāriti (kreiso parīti) (students, biļete) no studentiem, kuri gatavojas eksāmenam un sež telpā ar biļetem.

print("Atbild " + str(pair[0]) + ". students ar biļeti " + str(pair[1]) + ".") # Izvadam lietotājam informāciju, kurš students atbild un ar kādu biļeti.

exam\_tickets.add(pair[1]) # Pievienojam biļeti no korteža eksāmena biļetes kaudzītei (studenta biļeti, kurš ir atbildējis uz jautājumu).

print("Biļete", pair[1], "tiek atlikta biļešu \"kaudzē\".\n") # Izvadam informāciju, kura biļete tika atlikta kaudzē.

print\_pairs(students\_prepare) # Glīti izvadīt informāciju lietotājam par to, kuri cilvēki sēž un gatavojas eksāmenam iekšā telpā.

else:

atnac = choose\_random\_pair(students\_prepare, students, exam\_tickets) # Ja nav visi studenti ir ienākuši eksāmenā telpā, tad vajag aicināt nākamos (atnac) studentus uz eksāmenu (jo tie ir palikuši un vēl nenokārtoja eksāmenu). Nejauši izvelāmies studentu un viņam biļēti, no tiem, kuri vēl nav nokārtojuši.

print("Atnāc " + str(atnac[0]) + ". students un izvilka biļeti " + str(atnac[1]) + ".") # Izvadam lietotājam informāciju, kurš students atnāca un kādu biļeti paņēma.

students\_prepare.append(atnac) # Pievienojam sarakstam ar kortēžiem students\_prepare [(student, exam\_ticket), (student, exam\_ticket), ... , (student, exam\_ticket)] jauno kortēžu, ar studentu kurš atnāca un izvēlējas biļeti.

print\_pairs(students\_prepare) # Glīti izvadīt informāciju lietotājam par to, kuri cilvēki sēž un gatavojas eksāmenam iekšā telpā.

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

# Galvenā programmas daļa

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

students = set() # Tukša kopa ar visiem studentiem, tā tiek piepildīta ar str numuriem vēlāk šeit "convert\_int\_to\_str\_in\_set\_in\_range(1, 55, students)".

exam\_tickets = set() # Tukša kopa ar visiem eksāmena biļētem (kaudzē ar biļetem). Tā tiek piepildīta ar str burtiem no A līdz Z šeit "exam\_tickets\_words\_in\_range(65, 91, exam\_tickets)".

students\_prepare = [] # Tukšais saraksts, kur glābāsies visi kortēži [(studentu, biļeti), (studentu, biļeti), ... (studentu, biļeti)]. Tas tiek piepildīts ar start\_students\_number.

convert\_int\_to\_str\_in\_set\_in\_range(1, 55, students) # Lai visi skaitļi kopā, būtu uztvērti ka simbolu virknes (str). Citādi pop funkcijas nestrādās pareizi.

exam\_tickets\_words\_in\_range(65, 91, exam\_tickets) # Izveido biļetes numurus no A līdz Z. Tā tiek piepildīta ar str burtiem no A līdz Z.

# print("Studentu kopa:", students) # TESTĒŠANAI.

# print("Eksāmena biļešu kopa:", exam\_tickets) # TESTĒŠANAI.

start\_students\_number(10, students, exam\_tickets) # Izveido sakotnēju studentu sarakstu ar [(studentu, biļeti), (studentu, biļeti), ... (studentu, biļeti)].

print("Eksāmena simulācija.\nFormāts: ([Studenta numurs]-[Izvilkta biļete])\n") # Paskaidrojums lietotājam.

print("Pirmie desmit studenti ar izvilktam biļētem:") # Informācija lietotājam.

print\_pairs(students\_prepare) # Glīti izvada pirmus 10 studentus.

exam\_simulation(students\_prepare) # Sākt eksāmenu.

**Testa piemēri:**

1) Testa piemēri ir lieli, tāpēc viņi ir paradīti tikai daļēji.

Text

Description automatically generated

2)

Text

Description automatically generated

3)

Text

Description automatically generated

**2. uzdevums**

Sastādīt programmu, kas izveido klasi Taisnstūris, kas satur laukus - garums un platums un metodes - laukums un perimetrs. Izmantojot šo klasi izveidojiet objektus un nodrukājiet šo objektu laukumus un perimetrus.

**Kods: (klase Check ir uzrakstītas funkcijas, kas šajā kodā netiek izmantotas, tas ir tāpēc, ka es gribēju izmantot šo klasi arī citur)**

# Programmas nosaukums: Klase taisnstūris

# 2. uzdevums (1MPR11\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas izveido klasi Taisnstūris, kas satur laukus - garums un platums

# un metodes - laukums un perimetrs. Izmantojot šo klasi izveidojiet objektus un nodrukājiet šo objektu laukumus un perimetrus.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

class Taisnsturis:

# Klase taisnstūra laukuma un perimetra aprēķināšanos.

def \_\_init\_\_(self, garums, platums):

self.garums = garums

self.platums = platums

def laukums(self):

# Aprēķina taisnstūra laukumu, reizinot tā garumu ar tā platumu.

return self.garums \* self.platums

def perimetrs(self):

# Aprēķina taisnstūra perimetru, saskaitot garumu un platumu un reizinot ar 2.

return (self.garums + self.platums) \* 2

class Check:

# Klase ar visam funkcijam, kuri ir noderīgi ievaddates pārbaudei.

@staticmethod

def is\_real(value):

# Pārbauda vai simbolu virkne (value) ir reāls skaitlis no (-inf, +inf).

# value - str, simbolu virkne, kurā reprezente skaitļi.

try:

float(value)

return True

except:

return False

@staticmethod

def is\_real\_positive(value):

# Pārbauda vai simbolu virkne (value) ir reāls pozitīvs skaitlis no (0, +inf).

# value - str, simbolu virkne, kurā reprezente skaitļi.

try:

value = float(value)

if value > 0:

return True

else:

return False

return True

except:

return False

@staticmethod

def is\_real\_positive\_or\_zero(value):

# Pārbauda vai simbolu virkne (value) ir reāls pozitīvs skaitlis vai nulle [0, +inf).

# value - str, simbolu virkne, kurā reprezente skaitļi.

try:

value = float(value)

if value >= 0:

return True

else:

return False

return True

except:

return False

@staticmethod

def is\_natural(value):

# Pārbauda vai simbolu virkne (value) ir naturāls skaitlis 1, 2, 3, 4, 5, ...

# value - str, simbolu virkne, kurā reprezente skaitļi.

try:

value = int(value)

if value > 0:

return True

else:

return False

except:

return False

@staticmethod

def is\_whole(value):

# Pārbauda vai simbolu virkne (value) ir vesels skaitlis ... -3, -2, -1, 0, 1, 2, 3 ...

# value - str, simbolu virkne, kurā reprezente skaitļi.

try:

value = int(value)

if value >= 0:

return True

else:

return False

except:

return False

@staticmethod

def is\_even(x):

# Pārbauda vai simbolu virkne (value) ir pāra skaitlis 2n.

# x - str, simbolu virkne, kurā reprezente skaitļi.

try:

x = int(x)

if x % 2 == 0:

return True

else:

return False

except:

return False

@staticmethod

def is\_odd(x):

# Pārbauda vai simbolu virkne (value) ir nepāra skaitlis 2n + 1.

# x - str, simbolu virkne, kurā reprezente skaitļi.

try:

x = int(x)

if x % 2 != 0:

return True

else:

return False

except:

return False

@staticmethod

def is\_prime(n):

# Pārbauda vai simbolu virkne (value) ir pirmskaitlis.

# n - str, simbolu virkne, kurā reprezente skaitļi.

try:

n = int(n)

except:

return False

else:

if n <= 1:

return False

for i in range(2, n):

if (n % i) == 0:

return False

return True

def ievade(description, error\_desciption):

# Metode skaitļa ievādei. Jāizmanto tāda veidā: n = Check.ievade("Ievadiet --- ", "Kļūda! ---") (nevajag rakstīt ==>).

# description - str, simbolu virkne, kurā tiek parādīta, kad tiek paprasīts ievādit skaitļi.

# error\_desciption - str, simbolu virkne, kurā tiek parādīta, kad tiek paziņota kļūda.

n = input(description + "==> ")

while True:

if Check.is\_real\_positive(n): # is\_real(n): # Check.(funkcija, kas pārbauda nepieciešamo)

return float(n)

print(error\_desciption)

n = input(description + "==> ")

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

# Galvenā programmas daļa

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

n = Check.ievade("Ievadiet taisnstūra garumu ", "Kļūda! Ievadītai vērtībai jābūt reālam pozitīvam skaitlim!")

m = Check.ievade("Ievadiet taisnstūra platumu ", "Kļūda! Ievadītai vērtībai jābūt reālam pozitīvam skaitlim!")

taisnsturis = Taisnsturis(n, m)

print("\nTaisnstūra perimetrs. P =", taisnsturis.perimetrs())

print("Taisnstūra laukums. S =", taisnsturis.laukums())

**Testa piemēri:**

1)

Text

Description automatically generated

2)

Text

Description automatically generated

3)

Text

Description automatically generated

4)

Text

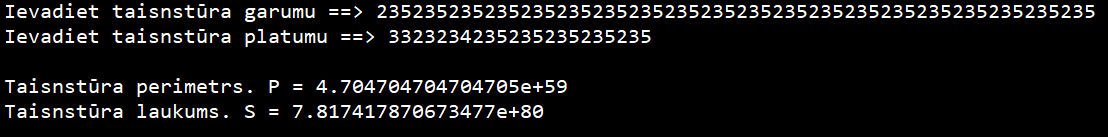
Description automatically generated

5)

Graphical user interface, text

Description automatically generated

6)



7)

Text

Description automatically generated

**3. uzdevums**

Sastādīt programmu, kas izveido klasi Trijstūris, kas satur 3 laukus, kas glabā trijstūra malu garumus un metodes - laukums, perimetrs, ievilktā riņķa rādiuss un apvilktā riņķa rādiuss, izmantojot šo klasi izveidojiet objektus un nodrukājiet šo objektu laukumus, perimetrus un ievilktās un apvilktās riņķa rādiusus.

**Kods: (klase Check ir uzrakstītas funkcijas, kas šajā kodā netiek izmantotas, tas ir tāpēc, ka es gribēju izmantot šo klasi arī citur)**

# Programmas nosaukums: Klase trijstūris

# 3. uzdevums (1MPR11\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas izveido klasi Trijstūris, kas satur 3 laukus, kas glabā trijstūra malu garumus

# un metodes - laukums, perimetrs, ievilktā riņķa rādiuss un apvilktā riņķa rādiuss, izmantojot šo klasi

# izveidojiet objektus un nodrukājiet šo objektu laukumus, perimetrus un ievilktās un apvilktās riņķa rādiusus.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

import math

class Trijsturis:

# Trijstūra klase.

def \_\_init\_\_(self, a, b, c):

# Iniciālizē a, b, c.

# a - trijstūra pirmās malas garums.

# b - trijstūra otrais malas garums.

# c - trijstūra trešais malas garums.

# self - trijstūris.

self.a = a

self.b = b

self.c = c

def pusperimetrs(self):

# Aprēķina trijstūra pusperimetru.

p = (self.a + self.b + self.c) / 2

return p

def laukums(self):

# Aprēķina trijstūra laukumu ar Herona formulu.

p = Trijsturis.pusperimetrs(self)

return math.sqrt(p \* (p - self.a) \* (p - self.b) \* (p - self.c)) # math.sqrt(p \* (p - self.a) \* (p - self.b) \* (p - self.c)) # self.garums \* self.platums

def perimetrs(self):

# Aprēķina trijstūra perimetru.

return self.a + self.b + self.c

def ievilkta\_rinka\_radiuss(self):

# Aprēķina trijstūra ievilkta riņķa līnijas rādiusu r.

return Trijsturis.laukums(self) / Trijsturis.pusperimetrs(self) # regulāram r = (a\*sqrt(3))/6

def apvilkta\_rinka\_radiuss(self):

# Aprēķina trijstūra apvilkta riņķa līnijas rādiusu R.

return self.a \* self.b \* self.c / (4 \* Trijsturis.laukums(self)) # regulāram R = (a\*sqrt(3))/6

class Check:

# Klase ar visam funkcijam, kuri ir noderīgi ievaddates pārbaudei.

@staticmethod

def is\_real(value):

# Pārbauda vai simbolu virkne (value) ir reāls skaitlis no (-inf, +inf).

# value - str, simbolu virkne, kurā reprezente skaitļi.

try:

float(value)

return True

except:

return False

@staticmethod

def is\_real\_positive(value):

# Pārbauda vai simbolu virkne (value) ir reāls pozitīvs skaitlis no (0, +inf).

# value - str, simbolu virkne, kurā reprezente skaitļi.

try:

value = float(value)

if value > 0:

return True

else:

return False

return True

except:

return False

@staticmethod

def is\_real\_positive\_or\_zero(value):

# Pārbauda vai simbolu virkne (value) ir reāls pozitīvs skaitlis vai nulle [0, +inf).

# value - str, simbolu virkne, kurā reprezente skaitļi.

try:

value = float(value)

if value >= 0:

return True

else:

return False

return True

except:

return False

@staticmethod

def is\_natural(value):

# Pārbauda vai simbolu virkne (value) ir naturāls skaitlis 1, 2, 3, 4, 5, ...

# value - str, simbolu virkne, kurā reprezente skaitļi.

try:

value = int(value)

if value > 0:

return True

else:

return False

except:

return False

@staticmethod

def is\_whole(value):

# Pārbauda vai simbolu virkne (value) ir vesels skaitlis ... -3, -2, -1, 0, 1, 2, 3 ...

# value - str, simbolu virkne, kurā reprezente skaitļi.

try:

value = int(value)

if value >= 0:

return True

else:

return False

except:

return False

@staticmethod

def is\_even(x):

# Pārbauda vai simbolu virkne (value) ir pāra skaitlis 2n.

# x - str, simbolu virkne, kurā reprezente skaitļi.

try:

x = int(x)

if x % 2 == 0:

return True

else:

return False

except:

return False

@staticmethod

def is\_odd(x):

# Pārbauda vai simbolu virkne (value) ir nepāra skaitlis 2n + 1.

# x - str, simbolu virkne, kurā reprezente skaitļi.

try:

x = int(x)

if x % 2 != 0:

return True

else:

return False

except:

return False

@staticmethod

def is\_prime(n):

# Pārbauda vai simbolu virkne (value) ir pirmskaitlis.

# n - str, simbolu virkne, kurā reprezente skaitļi.

try:

n = int(n)

except:

return False

else:

if n <= 1:

return False

for i in range(2, n):

if (n % i) == 0:

return False

return True

@staticmethod

def can\_make\_triangle(a, b, c):

# Pārbauda vai no šiem skaitļiem var izveidot trijstūri.

# a - float skaitlis intervāla (0, +inf)

# b - float skaitlis intervāla (0, +inf)

# c - float skaitlis intervāla (0, +inf)

if (a + b <= c) or (a + c <= b) or (b + c <= a):

return False

else:

return True

def ievade(description, error\_desciption):

# Metode skaitļa ievādei. Jāizmanto tāda veidā: n = Check.ievade("Ievadiet --- ", "Kļūda! ---") (nevajag rakstīt ==>).

# description - str, simbolu virkne, kurā tiek parādīta, kad tiek paprasīts ievādit skaitļi.

# error\_desciption - str, simbolu virkne, kurā tiek parādīta, kad tiek paziņota kļūda.

n = input(description + "==> ")

while True:

if Check.is\_real\_positive(n): # is\_real(n): # Check.(funkcija, kas pārbauda nepieciešamo)

return float(n)

print(error\_desciption)

n = input(description + "==> ")

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

# Galvenā programmas daļa

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

a = Check.ievade("Ievadiet trijstūra pirmas malas garumu ", "Kļūda! Ievadītai vērtībai jābūt reālam pozitīvam skaitlim!")

b = Check.ievade("Ievadiet trijstūra otras malas garumu ", "Kļūda! Ievadītai vērtībai jābūt reālam pozitīvam skaitlim!")

c = Check.ievade("Ievadiet trijstūra trešas malas garumu ", "Kļūda! Ievadītai vērtībai jābūt reālam pozitīvam skaitlim!")

if Check.can\_make\_triangle(a, b, c):

trijsturis = Trijsturis(a, b, c)

print("\nJūs ievadījāt trijstūri ar malas garumiem " + str(a) + ", " + str(b) + ", " + str(c))

print("\nTrijstūra perimetrs. P =", trijsturis.perimetrs())

print("Trijstūra laukums. S =", trijsturis.laukums())

print("Trijstūra ievilktas riņķa līnijas rādiuss. r =", trijsturis.ievilkta\_rinka\_radiuss())

print("Trijstūra apvilktas riņķa līnijas rādiuss. R =", trijsturis.apvilkta\_rinka\_radiuss())

else:

print("\nTrijstūris ar malam " + str(a) + ", " + str(b) + ", " + str(c) + " neeksistē!")

**Testa piemēri:**

1)

Text

Description automatically generated

2)

Graphical user interface, text, chat or text message

Description automatically generated

3)

Text

Description automatically generated

4)

Text

Description automatically generated

5)

Text

Description automatically generated

6)

Text

Description automatically generated

7)

Text

Description automatically generated

**4. uzdevums**

Sastādīt programmu, kas izveido klasi Kalkulators, kas satur 4 statiskās metodes četru aritmētisko darbību veikšanai. Ievadiet 2 skaitļus un veiciet ar tiem 4 aritmētiskās darbības, izmantojot iepriekš izveidotās klases statiskās metodes.

**Kods: (klase Check ir uzrakstītas funkcijas, kas šajā kodā netiek izmantotas, tas ir tāpēc, ka es gribēju izmantot šo klasi arī citur)**

# Programmas nosaukums: Kalkulators

# 4. uzdevums (1MPR11\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas izveido klasi Kalkulators, kas satur 4 statiskās metodes četru aritmētisko darbību veikšanai.

# Ievadiet 2 skaitļus un veiciet ar tiem 4 aritmētiskās darbības, izmantojot iepriekš izveidotās klases statiskās metodes.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

class Kalkulators:

# Kalkulatora klase

@staticmethod

def saskaitit(a, b):

# Saskaita a un b

# a - float skaitlis

# b - float skaitlis

return a + b

@staticmethod

def atnemt(a, b):

# Atņem a un b

# a - float skaitlis

# b - float skaitlis

return a - b

@staticmethod

def reizinat(a, b):

# Sareizina a un b

# a - float skaitlis

# b - float skaitlis

return a \* b

@staticmethod

def dalit(a, b):

# Izdala a ar b

# a - float skaitlis

# b - float skaitlis

if a == 0 and b == 0:

return "Nav definēts"

if a != 0 and b == 0:

return "Nedrīkst dalīt ar 0"

return a / b

class Check:

# Klase ar visam funkcijam, kuri ir noderīgi ievaddates pārbaudei.

@staticmethod

def is\_real(value):

# Pārbauda vai simbolu virkne (value) ir reāls skaitlis no (-inf, +inf).

# value - str, simbolu virkne, kurā reprezente skaitļi.

try:

float(value)

return True

except:

return False

@staticmethod

def is\_real\_positive(value):

# Pārbauda vai simbolu virkne (value) ir reāls pozitīvs skaitlis no (0, +inf).

# value - str, simbolu virkne, kurā reprezente skaitļi.

try:

value = float(value)

if value > 0:

return True

else:

return False

return True

except:

return False

@staticmethod

def is\_real\_positive\_or\_zero(value):

# Pārbauda vai simbolu virkne (value) ir reāls pozitīvs skaitlis vai nulle [0, +inf).

# value - str, simbolu virkne, kurā reprezente skaitļi.

try:

value = float(value)

if value >= 0:

return True

else:

return False

return True

except:

return False

@staticmethod

def is\_natural(value):

# Pārbauda vai simbolu virkne (value) ir naturāls skaitlis 1, 2, 3, 4, 5, ...

# value - str, simbolu virkne, kurā reprezente skaitļi.

try:

value = int(value)

if value > 0:

return True

else:

return False

except:

return False

@staticmethod

def is\_whole(value):

# Pārbauda vai simbolu virkne (value) ir vesels skaitlis ... -3, -2, -1, 0, 1, 2, 3 ...

# value - str, simbolu virkne, kurā reprezente skaitļi.

try:

value = int(value)

if value >= 0:

return True

else:

return False

except:

return False

@staticmethod

def is\_even(x):

# Pārbauda vai simbolu virkne (value) ir pāra skaitlis 2n.

# x - str, simbolu virkne, kurā reprezente skaitļi.

try:

x = int(x)

if x % 2 == 0:

return True

else:

return False

except:

return False

@staticmethod

def is\_odd(x):

# Pārbauda vai simbolu virkne (value) ir nepāra skaitlis 2n + 1.

# x - str, simbolu virkne, kurā reprezente skaitļi.

try:

x = int(x)

if x % 2 != 0:

return True

else:

return False

except:

return False

@staticmethod

def is\_prime(n):

# Pārbauda vai simbolu virkne (value) ir pirmskaitlis.

# n - str, simbolu virkne, kurā reprezente skaitļi.

try:

n = int(n)

except:

return False

else:

if n <= 1:

return False

for i in range(2, n):

if (n % i) == 0:

return False

return True

@staticmethod

def can\_make\_triangle(a, b, c):

# Pārbauda vai no šiem skaitļiem var izveidot trijstūri.

# a - float skaitlis intervāla (0, +inf)

# b - float skaitlis intervāla (0, +inf)

# c - float skaitlis intervāla (0, +inf)

if (a + b <= c) or (a + c <= b) or (b + c <= a):

return False

else:

return True

def ievade(description, error\_desciption):

# Metode skaitļa ievādei. Jāizmanto tāda veidā: n = Check.ievade("Ievadiet --- ", "Kļūda! ---") (nevajag rakstīt ==>).

# description - str, simbolu virkne, kurā tiek parādīta, kad tiek paprasīts ievādit skaitļi.

# error\_desciption - str, simbolu virkne, kurā tiek parādīta, kad tiek paziņota kļūda.

n = input(description + "==> ")

while True:

if Check.is\_real(n): # is\_real(n): # Check.(funkcija, kas pārbauda nepieciešamo)

return float(n)

print(error\_desciption)

n = input(description + "==> ")

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

# Galvenā programmas daļa

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

a = Check.ievade("Ievadiet pirmo skaitļi ", "Kļūda! Ievadītai vērtībai jābūt reālam skaitlim!")

b = Check.ievade("Ievadiet otro skaitļi ", "Kļūda! Ievadītai vērtībai jābūt reālam skaitlim!")

iekava1\_l = ""

iekava1\_r = ""

iekava2\_l = ""

iekava2\_r = ""

if a < 0:

iekava1\_l = "("

iekava1\_r = ")"

if b < 0:

iekava2\_l = "("

iekava2\_r = ")"

print()

print(iekava1\_l + str(a) + iekava1\_r + " + " + iekava2\_l + str(b) + iekava2\_r + " =", Kalkulators.saskaitit(a, b))

print(iekava1\_l + str(a) + iekava1\_r + " - " + iekava2\_l + str(b) + iekava2\_r + " =", Kalkulators.atnemt(a, b))

print(iekava1\_l + str(a) + iekava1\_r + " \* " + iekava2\_l + str(b) + iekava2\_r + " =", Kalkulators.reizinat(a, b))

if Kalkulators.dalit(a, b) == "Nav definēts" or Kalkulators.dalit(a, b) == "Nedrīkst dalīt ar 0":

sym = ""

else:

sym = " ="

print(iekava1\_l + str(a) + iekava1\_r + " : " + iekava2\_l + str(b) + iekava2\_r + sym, Kalkulators.dalit(a, b))

**Testa piemēri:**

1)

Text

Description automatically generated with medium confidence

2)

Text

Description automatically generated

3)

Text

Description automatically generated

4)

Graphical user interface, text, chat or text message

Description automatically generated

5)

Text

Description automatically generated

6)

Text

Description automatically generated

7)

Text

Description automatically generated

**5. uzdevums**

Sastādīt programmu, kas realizē teātra biļešu iegādi līdzīgi kā "Biļešu paradīze" izmantojot grafisko saskarni un OOP pieeju uzdevuma atrisināšanai. Uz ekrāna ir redzama skatītāju zāle ar visām sēdvietām un ar peles klikšķi var izvēlēties vienu vai vairākas biļetes, vai arī atcelt iepriekšējo izvēli. Pēc pogas Pirkt piespiešanas izvēlētas sēdvietas kļūst nepieejamas.

**Kods:**

# Programmas nosaukums: "Biļešu paradīze" ar klasēm.

# 5. uzdevums (1MPR11\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas realizē teātra biļešu iegādi līdzīgi kā "Biļešu paradīze" izmantojot grafisko saskarni un OOP pieeju uzdevuma atrisināšanai.

# Uz ekrāna ir redzama skatītāju zāle ar visām sēdvietām un ar peles klikšķi var izvēlēties vienu vai vairākas biļetes,

# vai arī atcelt iepriekšējo izvēli. Pēc pogas Pirkt piespiešanas izvēlētas sēdvietas kļūst nepieejamas.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

import tkinter

import random

class Seat:

# Sēdvietu klase.

def \_\_init\_\_(self, theater, rinda, kolonna):

# Sēdvietu iniciālizēšana.

self.status = 0 # Pirmkārt iestāsim, ka visas sēdvietas ir brīvas.

if random.randint(0, 1) == 1: # Nejaušam sēdvietu izvietojumam. Nevienmēr būs 50%. Jo random ir katrai sēdvietai, vai nu 0 vai 1 un tāpēc nebūs precīzi 50% katru reizi. Atkarīgs no veiksmes (liekas, ka tas ir labāk, jo interesantāk).

self.status = 1 # Ja tiek izlozēts 1 (50% varbūtība), tad vieta ir aizņemta.

self.button = tkinter.Button(theater, bg="#A00000", text="╳", width=5, height=2) # Izkrāsojam vietu ar sarkanu krāsu un liksim ╳ simbolu.

else: # Ja netika izlozēts 1 (tad izlozēts 0), tad vieta nebūs aizņemta.

self.button = tkinter.Button(theater, bg="green", text="○", width=5, height=2, command=self.change\_status) # Izkrāsojam vietu ar zaļu krāsu un liksim ○ simbolu.

self.button.grid(row=rinda, column=kolonna) # Definēsim, kur likt pogu.

def change\_status(self):

# Izmaina pogas stātusu no 0 -> 2, no 2 -> 0, no 1 -> 1.

# 0 - brīva vieta.

# 1 - aizņemta vieta.

# 2 - izvēlēta vieta.

if self.status == 0: # 0 -> 2 (brīva vieta uz izvelētu vietu)

self.status = 2

self.button.config(bg="yellow") # dzeltens reprezentē izvēlētu vietu.

elif self.status == 2: # 2 -> 0 (izvelēta uz brīvu, atcelt)

self.status = 0

self.button.config(bg="green", text="○") # zaļš reprezentē brīvo vieta.

elif self.status == 1: # Aizņemts. To nevar izmainīt. 1 -> 1.

pass

def occupy(self):

# Komanda sēdvietas aizņemšanai.

self.status = 1 # Pogas (vietas) statuss ir 1 (aizņemts).

self.button.configure(bg="#A00000", text="╳") # Izmainām krāsu uz sarkanu un izmainām tekstu uz pogas uz "╳".

class Teatris:

# Teātra klase.

def \_\_init\_\_(self, theater, rindas, kolonnas):

self.seats = []

self.selected\_seats = [] # Izvelētas sēdvietas saraksts.

self.num\_occupied\_seats = 0 # Nepieciešams, lai sekot līdzi aizņemto sēdvietu skaitam.

for i in range(rindas): # Izmantojot pilno pārlasi pārbaudam, vai visas vietas nav aizņemtas (varbūt ka "paveicas" un uzreiz viss ir izpārdots).

rinda = []

for j in range(kolonnas):

seat = Seat(theater, i, j)

if seat.status == 1:

self.num\_occupied\_seats += 1 # Ja sēdvieta sākotnēji ir aizņemta, tad palieliniet skaitītāju.

rinda.append(seat)

self.seats.append(rinda)

self.buy\_button = tkinter.Button(theater, text="Nopirkt izvēlētās biļetes", command=self.buy\_tickets) # Nopirkšanas poga.

self.buy\_button.grid(row=rindas + 1, column=0, columnspan=kolonnas)

self.all\_seat\_quantity = len(self.seats) \* len(self.seats[0]) # Lai zinātu cik ir kopējais sēdvietu skaits.

self.check\_occupied() # Izsaucam check\_occupied, lai atjauninātu nopirkšanas pogas stāvokli (ja visas ir aizņemtas vietas, tad bloķēt pogu).

def read\_selection(self):

# Atgriež sarakstu ar izvēlētam sēdvietam izmantojot pilnu pārlasi caur kātru vietu (rindu un kolonnu).

selected\_seats = [] # Tukšais saraksts ar izvēlētam sēdvietam.

for row in self.seats: # Iterējam caur katru rindu un kolonnu (pilnā pārlase)

for seat in row:

if seat.status == 2: # Un izmantojot pilno pārlasi, ja sēdvietai ir piekārtots 2 (izvelēta), tad .append sēdvietu sarakstam selected\_seats (izvēlētas sēdvietas)

selected\_seats.append(seat) # Pievienojam sarakstam atrasto izvēlēto vietu.

return selected\_seats # Atgriezt izvēlētas sēdvietas.

def buy\_tickets(self):

# Komanda pirkšanas pogai. Izsauc seat.occupy() visiem sēdvietam kas ie izvēlētas un palielina num\_occupied\_seats par katru izvēlēto vietu (tas ir nepieciešāms, lai jā visas vietās ir aizņemtas, tad bloķēt pogu).

self.selected\_seats = self.read\_selection() # Saraksts ar visam izvēlētam sēdvietām.

for seat in self.selected\_seats: # Izmainām vērtības visam izvēlētam vietam (izmainām 2 -> 1) un palielinām self.num\_occupied\_seats par vienu.

seat.occupy() # Izmainīt sēdvietas stavokļi uz ieņemtu.

self.num\_occupied\_seats += 1 # Palielinam aizņemto vietu skaitītāju par katru tikko aizņemto vietu.

self.check\_occupied() # Izsaukt check\_occupied, lai atjauninātu etiķetes un pogas stāvokli.

def check\_occupied(self):

# Komanda pārbauda vai visas vietas jau ir izpārdotas. Ja ir, tad bloķē pirkšanas pogu.

if self.num\_occupied\_seats == self.all\_seat\_quantity: # Ja visas vietas ir aizņemtas, tad nobloķēt pirkšanas pogu. Salīdzina aizņemto vietu skaitītāju, ar visu sēdvietu skaitu. Ja sakrīt tad, bloķēt pogu.

self.buy\_button.config(text="Visas vietas izpārdotas!", state="disabled") # Bloķēt pirkšanas pogu un izmainīt tekstu uz pogas.

class Define\_Theater:

# Klase teātra definēšanai.

@staticmethod

def izveidot\_teatri():

# Metode, kas nolasa ievādītas rindas un kolonnu vērtības no entry, nodzēs palīglogu teātra definēšanai, un izveido "zale" windows un izsauc klasi Teatris.

rindas = int(rindas\_entry.get()) # Nolasam vērtības no rindas entry.

kolonnas = int(kolonnas\_entry.get()) # Nolasam vērtības no kolonnas (sēdvietas) entry.

logs.destroy() # Nodzēsam logu, kur mēs prāsījam lietotājam ievādīt rindas un sēdvietu skaitu.

zale = tkinter.Tk() # Izveidojam jauno logu teātrim.

zale.title("Zāle") # Jauna loga virsraksts.

Teatris(zale, rindas, kolonnas) # Izsauc Teātra klasi ar lietotāja ievādītam rindam un kolonnam.

zale.mainloop() # Lai jauns logs būtu redzāms.

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

# Galvenā programmas daļa

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

# Izveidojam logu, kur paprasām lietotājam ievādīt rindas un kolonnu (sēdvietu) skaitu teātrī. Tas ir nepieciešams teātra izveidošanai.

logs = tkinter.Tk() # Faktiski tas ir papildlogs, lai lietotājs varētu ievādit rindas un kolonnu skaitu teātrī.

logs.title("Biļešu paradīze") # Loga virsraksts.

logs.geometry("250x125") # Loga izmērs.

rindas\_label = tkinter.Label(logs, text="Rindu skaits:") # Etiķete, lai lietotājs zinātu, ka jāievāda rindu skaitu.

rindas\_label.pack()

rindas\_entry = tkinter.Entry(logs) # Entry (ievaddlaukums) lietotājam rindas ievadei.

rindas\_entry.pack()

kolonas\_label = tkinter.Label(logs, text="Sēdvietu skaits:") # Etiķete, lai lietotājs zinātu, ka jāievāda sēdvietu (kolonnu) skaitu.

kolonas\_label.pack()

kolonnas\_entry = tkinter.Entry(logs) # Entry (ievaddlaukums) lietotājam sēdvietu (kolonnas) ievadei.

kolonnas\_entry.pack()

poga\_izveidot = tkinter.Button(logs, text="Izveidot teātru", command=Define\_Theater.izveidot\_teatri) # Poga "Izveidot teātru" papildlogam. Izsauc komandi "Define\_Theater.izveidot\_teatri"

poga\_izveidot.pack()

logs.mainloop() # Lai logs būtu redzāms.

**Testa piemēri:**

1) Izveidots papildus logs, lai lietotājs varētu ievadīt teātra izmēru. Kopēju rindu skaitu un sēdvietu skaitu.

Graphical user interface, application

Description automatically generated

2) Ievadam rindas un sēdvietu skaitu.

Graphical user interface, application

Description automatically generated

3) Tiek izveidots jauns logs (iepriekšējais tiek nodzēsts) kur aptuveni 50% ir jau aizņemti (varētu būt mazāk, vai lielāk, jo programmā katrai sēdvietai ir 50% varbūtība, ka tā būs aizņemta. Tas negarantē, ka katru reizi būs 50%. Atkarīgs no veiksmes.

Graphical user interface, application

Description automatically generated

4) Ar sarkanu – aizņemtas vietas. Ar zaļu – brīvas vietas. Ar dzelteno – ar peles klikšķi izvēlētas vietas.

Graphical user interface, application, table

Description automatically generated

5) Uzklikšķinot uz “Nopirkt izvēlētas biļetes” izvēlētas sēdvietas tiek izmainītas uz aizņemtam.

Graphical user interface, application, table

Description automatically generated

6) Vai izvēlēties lielu skaitu vienlaikus un nopirkt.

Shape

Description automatically generated with low confidence

7)

Table

Description automatically generated

8)

A picture containing table

Description automatically generated

9) Var izpirkt visu un tad pogai teksts izmainīsies uz “Visas vietas izpārdotas!” un poga tiek bloķēta.

Table

Description automatically generated

10) Ja tiek izveidots teātris 1x1, tad ja vieta ir aizņemta, tad uzreiz poga būs bloķēta un teksts būs “Visas vietas izpārdotas!”.

Application

Description automatically generated with low confidence

11)

A picture containing shape

Description automatically generated

12)

Diagram

Description automatically generated with medium confidence

13)

Chart

Description automatically generated

**PU1. uzdevums. Teātris papildus.**

Papildināt 5. Uzdevumā realizēto Teātra uzdevumu.

Teātris tika papildināts ar to, ka tagad var izvēlēties teātra izmērus sakum izveidotāja papildus logā, kurš tiek dzēsts, kad lietotājs izveido teātri. Programma ir papildināta ar pārbaudi vai visas vietas ir aizņemtas. Ja visas ir aizņemtas, tad nopirkšanas poga tiek bloķēta teksts uz pogas tiek izmainīts uz “Visas vietas izpārdotas!”. Grafiski tiek veiktas šādas izmaiņas: uz pogām ir X (aizņemts) vai O (brīvs vai izvēlēts), krāsas tika izmainītas uz sarkanu – aizņemtu , zaļu – brīvu, dzeltenais palika par izvēlēto.

**Kods (Ir tāds pats, kā 5.uzdevumā):**

# Programmas nosaukums: "Biļešu paradīze" ar klasēm.

# PU1. uzdevums (1MPR11\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas realizē teātra biļešu iegādi līdzīgi kā "Biļešu paradīze" izmantojot grafisko saskarni un OOP pieeju uzdevuma atrisināšanai.

# Uz ekrāna ir redzama skatītāju zāle ar visām sēdvietām un ar peles klikšķi var izvēlēties vienu vai vairākas biļetes,

# vai arī atcelt iepriekšējo izvēli. Pēc pogas Pirkt piespiešanas izvēlētas sēdvietas kļūst nepieejamas.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

"""

Teātris tika papildināts ar to, ka tagad var izvēlēties teātra izmērus sakum izveidotāja papildus logā,

kurš tiek dzēsts, kad lietotājs izveido teātri. Programma ir papildināta ar pārbaudi vai visas vietas ir aizņemtas.

Ja visas ir aizņemtas, tad nopirkšanas poga tiek bloķēta teksts uz pogas tiek izmainīts uz “Visas vietas izpārdotas!”.

Grafiski tiek veiktas šādas izmaiņas: uz pogām ir X (aizņemts) vai O (brīvs vai izvēlēts),

krāsas tika izmainītas uz sarkanu – aizņemtu , zaļu – brīvu, dzeltenais palika par izvēlēto.

"""

import tkinter

import random

class Seat:

# Sēdvietu klase.

def \_\_init\_\_(self, theater, rinda, kolonna):

# Sēdvietu iniciālizēšana.

self.status = 0 # Pirmkārt iestāsim, ka visas sēdvietas ir brīvas.

if random.randint(0, 1) == 1: # Nejaušam sēdvietu izvietojumam. Nevienmēr būs 50%. Jo random ir katrai sēdvietai, vai nu 0 vai 1 un tāpēc nebūs precīzi 50% katru reizi. Atkarīgs no veiksmes (liekas, ka tas ir labāk, jo interesantāk).

self.status = 1 # Ja tiek izlozēts 1 (50% varbūtība), tad vieta ir aizņemta.

self.button = tkinter.Button(theater, bg="#A00000", text="╳", width=5, height=2) # Izkrāsojam vietu ar sarkanu krāsu un liksim ╳ simbolu.

else: # Ja netika izlozēts 1 (tad izlozēts 0), tad vieta nebūs aizņemta.

self.button = tkinter.Button(theater, bg="green", text="○", width=5, height=2, command=self.change\_status) # Izkrāsojam vietu ar zaļu krāsu un liksim ○ simbolu.

self.button.grid(row=rinda, column=kolonna) # Definēsim, kur likt pogu.

def change\_status(self):

# Izmaina pogas stātusu no 0 -> 2, no 2 -> 0, no 1 -> 1.

# 0 - brīva vieta.

# 1 - aizņemta vieta.

# 2 - izvēlēta vieta.

if self.status == 0: # 0 -> 2 (brīva vieta uz izvelētu vietu)

self.status = 2

self.button.config(bg="yellow") # dzeltens reprezentē izvēlētu vietu.

elif self.status == 2: # 2 -> 0 (izvelēta uz brīvu, atcelt)

self.status = 0

self.button.config(bg="green", text="○") # zaļš reprezentē brīvo vieta.

elif self.status == 1: # Aizņemts. To nevar izmainīt. 1 -> 1.

pass

def occupy(self):

# Komanda sēdvietas aizņemšanai.

self.status = 1 # Pogas (vietas) statuss ir 1 (aizņemts).

self.button.configure(bg="#A00000", text="╳") # Izmainām krāsu uz sarkanu un izmainām tekstu uz pogas uz "╳".

class Teatris:

# Teātra klase.

def \_\_init\_\_(self, theater, rindas, kolonnas):

self.seats = []

self.selected\_seats = [] # Izvelētas sēdvietas saraksts.

self.num\_occupied\_seats = 0 # Nepieciešams, lai sekot līdzi aizņemto sēdvietu skaitam.

for i in range(rindas): # Izmantojot pilno pārlasi pārbaudam, vai visas vietas nav aizņemtas (varbūt ka "paveicas" un uzreiz viss ir izpārdots).

rinda = []

for j in range(kolonnas):

seat = Seat(theater, i, j)

if seat.status == 1:

self.num\_occupied\_seats += 1 # Ja sēdvieta sākotnēji ir aizņemta, tad palieliniet skaitītāju.

rinda.append(seat)

self.seats.append(rinda)

self.buy\_button = tkinter.Button(theater, text="Nopirkt izvēlētās biļetes", command=self.buy\_tickets) # Nopirkšanas poga.

self.buy\_button.grid(row=rindas + 1, column=0, columnspan=kolonnas)

self.all\_seat\_quantity = len(self.seats) \* len(self.seats[0]) # Lai zinātu cik ir kopējais sēdvietu skaits.

self.check\_occupied() # Izsaucam check\_occupied, lai atjauninātu nopirkšanas pogas stāvokli (ja visas ir aizņemtas vietas, tad bloķēt pogu).

def read\_selection(self):

# Atgriež sarakstu ar izvēlētam sēdvietam izmantojot pilnu pārlasi caur kātru vietu (rindu un kolonnu).

selected\_seats = [] # Tukšais saraksts ar izvēlētam sēdvietam.

for row in self.seats: # Iterējam caur katru rindu un kolonnu (pilnā pārlase)

for seat in row:

if seat.status == 2: # Un izmantojot pilno pārlasi, ja sēdvietai ir piekārtots 2 (izvelēta), tad .append sēdvietu sarakstam selected\_seats (izvēlētas sēdvietas)

selected\_seats.append(seat) # Pievienojam sarakstam atrasto izvēlēto vietu.

return selected\_seats # Atgriezt izvēlētas sēdvietas.

def buy\_tickets(self):

# Komanda pirkšanas pogai. Izsauc seat.occupy() visiem sēdvietam kas ie izvēlētas un palielina num\_occupied\_seats par katru izvēlēto vietu (tas ir nepieciešāms, lai jā visas vietās ir aizņemtas, tad bloķēt pogu).

self.selected\_seats = self.read\_selection() # Saraksts ar visam izvēlētam sēdvietam.

for seat in self.selected\_seats: # Izmainām vērtības visam izvēlētam vietam (izmainām 2 -> 1) un palielinām self.num\_occupied\_seats par vienu.

seat.occupy() # Izmainīt sēdvietas stavokļi uz ieņemtu.

self.num\_occupied\_seats += 1 # Palielinam aizņemto vietu skaitītāju par katru tikko aizņemto vietu.

self.check\_occupied() # Izsaukt check\_occupied, lai atjauninātu etiķetes un pogas stāvokli.

def check\_occupied(self):

# Komanda pārbauda vai visas vietas jau ir izpārdotas. Ja ir, tad bloķē pirkšanas pogu.

if self.num\_occupied\_seats == self.all\_seat\_quantity: # Ja visas vietas ir aizņemtas, tad nobloķēt pirkšanas pogu. Salidzina aizņemto vietu skaititāju, ar visu sēdvietu skaitu. Ja sakrīt tad, bloķēt pogu.

self.buy\_button.config(text="Visas vietas izpārdotas!", state="disabled") # Bloķēt pirkšanas pogu un izmainīt tekstu uz pogas.

class Define\_Theater:

# Klase teātra definēšanai.

@staticmethod

def izveidot\_teatri():

# Metode, kas nolasa ievādītas rindas un kolonnu vērtības no entry, nodzēs palīglogu teātra definēšanai, un izveido "zale" windows un izsauc klasi Teatris.

rindas = int(rindas\_entry.get()) # Nolasam vērtības no rindas entry.

kolonnas = int(kolonnas\_entry.get()) # Nolasam vērtības no kolonnas (sēdvietas) entry.

logs.destroy() # Nodzēsam logu, kur mēs prāsījam lietotājam ievādīt rindas un sēdvietu skaitu.

zale = tkinter.Tk() # Izveidojam jauno logu teātrim.

zale.title("Zāle") # Jauna loga virsraksts.

Teatris(zale, rindas, kolonnas) # Izsauc Teātra klasi ar lietotāja ievādītam rindam un kolonnam.

zale.mainloop() # Lai jauns logs būtu redzāms.

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

# Galvenā programmas daļa

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

# Izveidojam logu, kur paprasām lietotājam ievādīt rindas un kolonnu (sēdvietu) skaitu teātrī. Tas ir nepieciešams teātra izveidošanai.

logs = tkinter.Tk() # Faktiski tas ir papildlogs, lai lietotājs varētu ievādit rindas un kolonnu skaitu teātrī.

logs.title("Biļešu paradīze") # Loga virsraksts.

logs.geometry("250x125") # Loga izmērs.

rindas\_label = tkinter.Label(logs, text="Rindu skaits:") # Etiķete, lai lietotājs zinātu, ka jāievāda rindu skaitu.

rindas\_label.pack()

rindas\_entry = tkinter.Entry(logs) # Entry (ievaddlaukums) lietotājam rindas ievadei.

rindas\_entry.pack()

kolonas\_label = tkinter.Label(logs, text="Sēdvietu skaits:") # Etiķete, lai lietotājs zinātu, ka jāievāda sēdvietu (kolonnu) skaitu.

kolonas\_label.pack()

kolonnas\_entry = tkinter.Entry(logs) # Entry (ievaddlaukums) lietotājam sēdvietu (kolonnas) ievadei.

kolonnas\_entry.pack()

poga\_izveidot = tkinter.Button(logs, text="Izveidot teātru", command=Define\_Theater.izveidot\_teatri) # Poga "Izveidot teātru" papildlogam. Izsauc komandi "Define\_Theater.izveidot\_teatri"

poga\_izveidot.pack()

logs.mainloop() # Lai logs būtu redzams.

**Testa piemēri:**

1) Izveidots papildus logs, lai lietotājs varētu ievadīt teātra izmēru. Kopēju rindu skaitu un sēdvietu skaitu.

Graphical user interface, application

Description automatically generated

2) Ievadam rindas un sēdvietu skaitu.

Graphical user interface, application

Description automatically generated

3) Tiek izveidots jauns logs (iepriekšējais tiek nodzēsts) kur aptuveni 50% ir jau aizņemti (varētu būt mazāk, vai lielāk, jo programmā katrai sēdvietai ir 50% varbūtība, ka tā būs aizņemta. Tas negarantē, ka katru reizi būs 50%. Atkarīgs no veiksmes.

Graphical user interface, application

Description automatically generated

4) Ar sarkanu – aizņemtas vietas. Ar zaļu – brīvas vietas. Ar dzelteno – ar peles klikšķi izvēlētas vietas.

Graphical user interface, application, table

Description automatically generated

5) Uzklikšķinot uz “Nopirkt izvēlētas biļetes” izvēlētas sēdvietas tiek izmainītas uz aizņemtam.

Graphical user interface, application, table

Description automatically generated

6) Vai izvēlēties lielu skaitu vienlaikus un nopirkt.

Shape

Description automatically generated with low confidence

7)

Table

Description automatically generated

8)

A picture containing table

Description automatically generated

9) Var izpirkt visu un tad pogai teksts izmainīsies uz “Visas vietas izpārdotas!” un poga tiek bloķēta.

Table

Description automatically generated

10) Ja tiek izveidots teātris 1x1, tad ja vieta ir aizņemta, tad uzreiz poga būs bloķēta un teksts būs “Visas vietas izpārdotas!”.

Application

Description automatically generated with low confidence

11)

A picture containing shape

Description automatically generated

12)

Diagram

Description automatically generated with medium confidence

13)

Chart

Description automatically generated