

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 izvēlēti, 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 secībā studenti tika eksaminēti. Paziņot uz ekrāna, kādā secībā 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 izvēlēti, 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 secībā 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
    kortēžu 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 biļētem 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 divām vērtībām [(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 jābūt kopai ar eksāmena biļētiem no A līdz Z.
```

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

```
    # m - int, līdz cik (ASCII kods).
```

```
# (65, 91) - lielajiem latīņu 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 sastāv 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 pirmajiem.
```

```
    # 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 biļešu, 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 pāri (kreiso pāri) (students, biļete) no studentiem, kuri gatavojas eksāmenam un sež telpā ar biļetēm.

print("Atbild " + str(pair[0]) + ". students ar biļeti " + str(pair[1]) + ".") # Izvadām 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") # Izvadām informāciju, kura biļete tika atlikta kaudzē.

if len(students) == 0: # Ja visi studenti jau ir ienākuši eksāmenā telpā, tad jau mēs vairs nevienu neaicināsim atkāļ 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 pāri (kreiso pāri) (students, biļete) no studentiem, kuri gatavojas eksāmenam un sež telpā ar biļetēm.

print("Atbild " + str(pair[0]) + ". students ar biļeti " + str(pair[1]) + ".") # Izvadām 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") # Izvadām 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 nāamos (atnac) studentus uz eksāmenu (jo tie

ir palikuši un vēl nenokārtoja eksāmenu). Nejauši izvelāties studentu un viņam biļeti, no tiem, kuri vēl nav nokārtojuši.

```
print("Atnāc " + str(atnac[0]) + ". students un izvilka biļeti " + str(atnac[1]) + ".") # Izvadām 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 izvēlējās 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 biļetēm (kaudzē ar biļetēm). 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]-[Izvilktā 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 parādīti tikai daļēji.

```
Eksāmena simulācija.  
Formāts: ([Studenta numurs]-[Izvilкта biļete])  
  
Pirmie desmit studenti ar izvilktam biļētem:  
21-R, 14-J, 43-M, 17-S, 39-B, 44-Z, 25-E, 7-U, 49-A, 23-L  
Atbild 21. students ar biļeti R.  
Biļete R tiek atlikta biļešu "kaudzē".  
  
Atnāc 53. students un izvilka biļeti O.  
14-J, 43-M, 17-S, 39-B, 44-Z, 25-E, 7-U, 49-A, 23-L, 53-O  
Atbild 14. students ar biļeti J.  
Biļete J tiek atlikta biļešu "kaudzē".  
  
Atnāc 1. students un izvilka biļeti Q.  
43-M, 17-S, 39-B, 44-Z, 25-E, 7-U, 49-A, 23-L, 53-O, 1-Q  
Atbild 43. students ar biļeti M.  
Biļete M tiek atlikta biļešu "kaudzē".  
  
Atnāc 20. students un izvilka biļeti G.  
17-S, 39-B, 44-Z, 25-E, 7-U, 49-A, 23-L, 53-O, 1-Q, 20-G  
Atbild 17. students ar biļeti S.  
Biļete S tiek atlikta biļešu "kaudzē".  
  
Atnāc 34. students un izvilka biļeti H.  
39-B, 44-Z, 25-E, 7-U, 49-A, 23-L, 53-O, 1-Q, 20-G, 34-H  
Atbild 39. students ar biļeti B.  
Biļete B tiek atlikta biļešu "kaudzē".  
  
Atnāc 45. students un izvilka biļeti W.  
44-Z, 25-E, 7-U, 49-A, 23-L, 53-O, 1-Q, 20-G, 34-H, 45-W  
Atbild 44. students ar biļeti Z.  
Biļete Z tiek atlikta biļešu "kaudzē".  
  
Atnāc 9. students un izvilka biļeti Y.  
25-E, 7-U, 49-A, 23-L, 53-O, 1-Q, 20-G, 34-H, 45-W, 9-Y  
Atbild 25. students ar biļeti E.  
Biļete E tiek atlikta biļešu "kaudzē".  
  
Atnāc 19. students un izvilka biļeti V.  
7-U, 49-A, 23-L, 53-O, 1-Q, 20-G, 34-H, 45-W, 9-Y, 19-V  
Atbild 7. students ar biļeti U.  
Biļete U tiek atlikta biļešu "kaudzē".  
  
Atnāc 46. students un izvilka biļeti K.  
49-A, 23-L, 53-O, 1-Q, 20-G, 34-H, 45-W, 9-Y, 19-V, 46-K  
Atbild 49. students ar biļeti A.  
Biļete A tiek atlikta biļešu "kaudzē".  
  
Atnāc 22. students un izvilka biļeti C.  
23-L, 53-O, 1-Q, 20-G, 34-H, 45-W, 9-Y, 19-V, 46-K, 22-C  
Atbild 23. students ar biļeti L.  
Biļete L tiek atlikta biļešu "kaudzē".  
  
Atnāc 51. students un izvilka biļeti P.  
53-O, 1-Q, 20-G, 34-H, 45-W, 9-Y, 19-V, 46-K, 22-C, 51-P  
Atbild 53. students ar biļeti O.  
Biļete O tiek atlikta biļešu "kaudzē".
```

2)

Atbild 28. students ar biļeti Y.

Biļete Y tiek atlikta biļešu "kaudzē".

Atnāc 11. students un izvilka biļeti G.

35-V, 26-K, 27-C, 12-P, 10-T, 54-D, 37-I, 3-X, 13-F, 11-G

Atbild 35. students ar biļeti V.

Biļete V tiek atlikta biļešu "kaudzē".

Atnāc 52. students un izvilka biļeti N.

26-K, 27-C, 12-P, 10-T, 54-D, 37-I, 3-X, 13-F, 11-G, 52-N

Atbild 26. students ar biļeti K.

Biļete K tiek atlikta biļešu "kaudzē".

Atnāc 8. students un izvilka biļeti R.

27-C, 12-P, 10-T, 54-D, 37-I, 3-X, 13-F, 11-G, 52-N, 8-R

Atbild 27. students ar biļeti C.

Biļete C tiek atlikta biļešu "kaudzē".

12-P, 10-T, 54-D, 37-I, 3-X, 13-F, 11-G, 52-N, 8-R

Atbild 12. students ar biļeti P.

Biļete P tiek atlikta biļešu "kaudzē".

10-T, 54-D, 37-I, 3-X, 13-F, 11-G, 52-N, 8-R

Atbild 10. students ar biļeti T.

Biļete T tiek atlikta biļešu "kaudzē".

54-D, 37-I, 3-X, 13-F, 11-G, 52-N, 8-R

Atbild 54. students ar biļeti D.

Biļete D tiek atlikta biļešu "kaudzē".

37-I, 3-X, 13-F, 11-G, 52-N, 8-R

Atbild 37. students ar biļeti I.

Biļete I tiek atlikta biļešu "kaudzē".

3-X, 13-F, 11-G, 52-N, 8-R

Atbild 3. students ar biļeti X.

Biļete X tiek atlikta biļešu "kaudzē".

13-F, 11-G, 52-N, 8-R

Atbild 13. students ar biļeti F.

Biļete F tiek atlikta biļešu "kaudzē".

11-G, 52-N, 8-R

Atbild 11. students ar biļeti G.

Biļete G tiek atlikta biļešu "kaudzē".

52-N, 8-R

Atbild 52. students ar biļeti N.

Biļete N tiek atlikta biļešu "kaudzē".

8-R

Atbild 8. students ar biļeti R.

Biļete R tiek atlikta biļešu "kaudzē".

Eksāmens beidzās!

3)

Atbild 50. students ar biļeti T.
Biļete T tiek atlikta biļešu "kaudzē".

Atnāc 33. students un izvilka biļeti Y.
49-X, 47-I, 11-P, 13-K, 30-F, 28-M, 3-U, 21-E, 2-Q, 33-Y
Atbild 49. students ar biļeti X.
Biļete X tiek atlikta biļešu "kaudzē".

Atnāc 6. students un izvilka biļeti L.
47-I, 11-P, 13-K, 30-F, 28-M, 3-U, 21-E, 2-Q, 33-Y, 6-L
Atbild 47. students ar biļeti I.
Biļete I tiek atlikta biļešu "kaudzē".

Atnāc 41. students un izvilka biļeti B.
11-P, 13-K, 30-F, 28-M, 3-U, 21-E, 2-Q, 33-Y, 6-L, 41-B
Atbild 11. students ar biļeti P.
Biļete P tiek atlikta biļešu "kaudzē".

13-K, 30-F, 28-M, 3-U, 21-E, 2-Q, 33-Y, 6-L, 41-B
Atbild 13. students ar biļeti K.
Biļete K tiek atlikta biļešu "kaudzē".

30-F, 28-M, 3-U, 21-E, 2-Q, 33-Y, 6-L, 41-B
Atbild 30. students ar biļeti F.
Biļete F tiek atlikta biļešu "kaudzē".

28-M, 3-U, 21-E, 2-Q, 33-Y, 6-L, 41-B
Atbild 28. students ar biļeti M.
Biļete M tiek atlikta biļešu "kaudzē".

3-U, 21-E, 2-Q, 33-Y, 6-L, 41-B
Atbild 3. students ar biļeti U.
Biļete U tiek atlikta biļešu "kaudzē".

21-E, 2-Q, 33-Y, 6-L, 41-B
Atbild 21. students ar biļeti E.
Biļete E tiek atlikta biļešu "kaudzē".

2-Q, 33-Y, 6-L, 41-B
Atbild 2. students ar biļeti Q.
Biļete Q tiek atlikta biļešu "kaudzē".

33-Y, 6-L, 41-B
Atbild 33. students ar biļeti Y.
Biļete Y tiek atlikta biļešu "kaudzē".

6-L, 41-B
Atbild 6. students ar biļeti L.
Biļete L tiek atlikta biļešu "kaudzē".

41-B
Atbild 41. students ar biļeti B.
Biļete B tiek atlikta biļešu "kaudzē".

Eksāmens beidzās!

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 nodrukājiēt 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 nodrukājiēt  
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 visām funkcijām, 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 reprezentē 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 reprezentē 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 reprezentē 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 reprezentē 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 reprezentē 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 reprezentē skaitļi.  
    try:  
        x = int(x)  
        if x % 2 == 0:  
            return True  
    except:  
        return False
```

```
@staticmethod
```

```
def is_odd(x):  
    # Pārbauda vai simbolu virkne (value) ir nepāra skaitlis  $2n + 1$ .  
    # x - str, simbolu virkne, kurā reprezentē skaitļi.  
    try:  
        x = int(x)  
        if x % 2 != 0:  
            return True  
    except:  
        return False
```

```
@staticmethod
```

```
def is_prime(n):
```

```
# Pārbauda vai simbolu virkne (value) ir pirmskaitlis.
```

```
# n - str, simbolu virkne, kurā reprezentē 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_description):
```

```
    # 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_description - 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_description)
```

```
            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)

```
Ievadiet taisnstūra garumu ==> 5
Ievadiet taisnstūra platumu ==> 5

Taisnstūra perimetrs. P = 20.0
Taisnstūra laukums. S = 25.0
```

2)

```
Ievadiet taisnstūra garumu ==> pieci
Kļūda! Ievadītai vērtībai jābūt reālam pozitīvam skaitlim!
Ievadiet taisnstūra garumu ==> 5
Ievadiet taisnstūra platumu ==> -3
Kļūda! Ievadītai vērtībai jābūt reālam pozitīvam skaitlim!
Ievadiet taisnstūra platumu ==> -3.5
Kļūda! Ievadītai vērtībai jābūt reālam pozitīvam skaitlim!
Ievadiet taisnstūra platumu ==> 5.5

Taisnstūra perimetrs. P = 21.0
Taisnstūra laukums. S = 27.5
```

3)

```
Ievadiet taisnstūra garumu ==> 3
Ievadiet taisnstūra platumu ==> 3
```

Taisnstūra perimetrs. $P = 12.0$
Taisnstūra laukums. $S = 9.0$

4)

```
Ievadiet taisnstūra garumu ==> -3
Kļūda! Ievadītai vērtībai jābūt reālam pozitīvam skaitlim!
Ievadiet taisnstūra garumu ==> 0
Kļūda! Ievadītai vērtībai jābūt reālam pozitīvam skaitlim!
Ievadiet taisnstūra garumu ==> 1
Ievadiet taisnstūra platumu ==> 1
```

Taisnstūra perimetrs. $P = 4.0$
Taisnstūra laukums. $S = 1.0$

5)

```
Ievadiet taisnstūra garumu ==> 5
Ievadiet taisnstūra platumu ==> 5.2532523523552
```

Taisnstūra perimetrs. $P = 20.5065047047104$
Taisnstūra laukums. $S = 26.266261761776$

6)

[illegible]

Taisnstūra perimetrs. $P = 4.704704704704705e+59$
Taisnstūra laukums. $S = 7.817417870673477e+80$

7)

```
Ievadiet taisnstūra garumu ==> 12.235235235235235235235235
Ievadiet taisnstūra platumu ==> 1.1512524
```

Taisnstūra perimetrs. $P = 26.77297527047047$
Taisnstūra laukums. $S = 14.085843929129128$

3. uzdevums

Sastādīt programmu, kas izveido klasi Trijstūris, kas satur 3 laukus, kas glabā trijstūra malu garumus un metodes - laukums, perimetrs, ievilkta riņķa rādiuss un apvilkta riņķa rādiuss, izmantojot šo klasi izveidojiet objektus un nodrukājiēt šo objektu laukumus, perimetrus un ievilkta un apvilkta 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, ievilkta riņķa rādiuss un apvilkta riņķa rādiuss, izmantojot šo klasi
```

```
# izveidojiet objektus un nodrukājiēt šo objektu laukumus, perimetrus un ievilkta un apvilkta riņķa rādiusus.
```

```
# Programmas autors: Vladislavs Babaņins
```

```
# Versija 1.0
```

```
import math
```

```
class Trijsturis:
```

```
    # Trijstūra klase.
```

```
    def __init__(self, a, b, c):
```

```
        # Inicializē 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 apvilktā_rinka_radiuss(self):
```

```
    # Aprēķina trijstūra apvilktā 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 visām funkcijām, 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 reprezentē 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 reprezentē 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 reprezentē 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 reprezentē 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 reprezentē 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 reprezentē 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 reprezentē 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 reprezentē 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_description):
    # 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_description - 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_description)
        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 ievilkta riņķa līnijas rādiuss. r =", trijsturis.ievilkta_rinka_radiuss())

    print("Trijstūra apvilkta riņķa līnijas rādiuss. R =", trijsturis.apvilkta_rinka_radiuss())

else:

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

```

Testa piemēri:

1)

```

Ievadiet trijstūra pirmas malas garumu ==> 3
Ievadiet trijstūra otras malas garumu ==> 4
Ievadiet trijstūra trešas malas garumu ==> 5

Jūs ievadījāt trijstūri ar malas garumiem 3.0, 4.0, 5.0

Trijstūra perimetrs. P = 12.0
Trijstūra laukums. S = 6.0
Trijstūra ievilkta riņķa līnijas rādiuss. r = 1.0
Trijstūra apvilkta riņķa līnijas rādiuss. R = 2.5

```

2)

```
Ievadiet trijstūra pirmas malas garumu ==> 1
Ievadiet trijstūra otras malas garumu ==> 626
Ievadiet trijstūra trešas malas garumu ==> 1

Trijstūris ar malām 1.0, 626.0, 1.0 neeksistē!
```

3)

```
Ievadiet trijstūra pirmas malas garumu ==> 12.1241
Ievadiet trijstūra otras malas garumu ==> 12
Ievadiet trijstūra trešas malas garumu ==> 122

Trijstūris ar malām 12.1241, 12.0, 122.0 neeksistē!
```

4)

```
Ievadiet trijstūra pirmas malas garumu ==> 2
Ievadiet trijstūra otras malas garumu ==> 2
Ievadiet trijstūra trešas malas garumu ==> 4

Trijstūris ar malām 2.0, 2.0, 4.0 neeksistē!
```

5)

```
Ievadiet trijstūra pirmas malas garumu ==> 3
Ievadiet trijstūra otras malas garumu ==> 3
Ievadiet trijstūra trešas malas garumu ==> 3

Jūs ievadījāt trijstūri ar malām 3.0, 3.0, 3.0

Trijstūra perimetrs. P = 9.0
Trijstūra laukums. S = 3.897114317029974
Trijstūra ievilkta riņķa līnijas rādiuss. r = 0.8660254037844387
Trijstūra apvilkta riņķa līnijas rādiuss. R = 1.7320508075688772
```


6)

```
Ievadiet trijstūra pirmas malas garumu ==> 4
Ievadiet trijstūra otras malas garumu ==> 7
Ievadiet trijstūra trešas malas garumu ==> 3.24

Jūs ievadījāt trijstūri ar malas garumiem 4.0, 7.0, 3.24

Trijstūra perimetrs. P = 14.24
Trijstūra laukums. S = 3.2160573129221453
Trijstūra ievilkta riņķa līnijas rādiuss. r = 0.451693442826144
Trijstūra apvilkta riņķa līnijas rādiuss. R = 7.052113128976766
```

7)

```
Ievadiet trijstūra pirmas malas garumu ==> pieci
Kļūda! Ievadītai vērtībai jābūt reālam pozitīvam skaitlim!
Ievadiet trijstūra pirmas malas garumu ==> -3
Kļūda! Ievadītai vērtībai jābūt reālam pozitīvam skaitlim!
Ievadiet trijstūra pirmas malas garumu ==> 0
Kļūda! Ievadītai vērtībai jābūt reālam pozitīvam skaitlim!
Ievadiet trijstūra pirmas malas garumu ==> -3.3
Kļūda! Ievadītai vērtībai jābūt reālam pozitīvam skaitlim!
Ievadiet trijstūra pirmas malas garumu ==> 3.3
Ievadiet trijstūra otras malas garumu ==> 5.2
Ievadiet trijstūra trešas malas garumu ==> 4

Jūs ievadījāt trijstūri ar malas garumiem 3.3, 5.2, 4.0

Trijstūra perimetrs. P = 12.5
Trijstūra laukums. S = 6.599893465049265
Trijstūra ievilkta riņķa līnijas rādiuss. r = 1.0559829544078823
Trijstūra apvilkta riņķa līnijas rādiuss. R = 2.600041968991375
```

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 visām funkcijām, kuras ir noderīgas ievaddašu pārbaudei.
```

```
@staticmethod
```

```
def is_real(value):
```

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

```
    # value - str, simbolu virkne, kurā reprezentē 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 reprezentē 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 reprezentē 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 reprezentē 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 reprezentē 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 reprezentē 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 reprezentē 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 reprezentē 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_description):
    # 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_description - 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_description)
        n = input(description + "==> ")

# -----
# Galvenā programmas daļa
# -----

```

```

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

```
Ievadiet pirmo skaitli ==> 3  
Ievadiet otro skaitli ==> 3
```

```
3.0 + 3.0 = 6.0  
3.0 - 3.0 = 0.0  
3.0 * 3.0 = 9.0  
3.0 : 3.0 = 1.0
```

2)

```
Ievadiet pirmo skaitli ==> 3  
Ievadiet otro skaitli ==> -4
```

```
3.0 + (-4.0) = -1.0  
3.0 - (-4.0) = 7.0  
3.0 * (-4.0) = -12.0  
3.0 : (-4.0) = -0.75
```

3)

```
Ievadiet pirmo skaitli ==> -5  
Ievadiet otro skaitli ==> -6
```

```
(-5.0) + (-6.0) = -11.0  
(-5.0) - (-6.0) = 1.0  
(-5.0) * (-6.0) = 30.0  
(-5.0) : (-6.0) = 0.83333333333333333334
```

4)

```
Ievadiet pirmo skaitli ==> 1
Ievadiet otro skaitli ==> 1

1.0 + 1.0 = 2.0
1.0 - 1.0 = 0.0
1.0 * 1.0 = 1.0
1.0 : 1.0 = 1.0
```

5)

```
Ievadiet pirmo skaitli ==> 0
Ievadiet otro skaitli ==> 0

0.0 + 0.0 = 0.0
0.0 - 0.0 = 0.0
0.0 * 0.0 = 0.0
0.0 : 0.0 Nav definēts
```

6)

```
Ievadiet pirmo skaitli ==> 1
Ievadiet otro skaitli ==> 0

1.0 + 0.0 = 1.0
1.0 - 0.0 = 1.0
1.0 * 0.0 = 0.0
1.0 : 0.0 Nedrīkst dalīt ar 0
```

7)

```
Ievadiet pirmo skaitli ==> pieci
Kļūda! Ievadītai vērtībai jābūt reālam skaitlim!
Ievadiet pirmo skaitli ==> labi
Kļūda! Ievadītai vērtībai jābūt reālam skaitlim!
Ievadiet pirmo skaitli ==> 5
Ievadiet otro skaitli ==> 5.523525

5.0 + 5.523525 = 10.523525
5.0 - 5.523525 = -0.5235250000000002
5.0 * 5.523525 = 27.617625
5.0 : 5.523525 = 0.9052190403772953
```

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 inicializēšana.
```

```
        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="X", width=5, height=2) # Izkrāsojam vietu ar sarkanu krāsu un liksim X simbolu.
```

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

```
self.button = tkinter.Button(theater, bg="green", text="o", width=5, height=2,
command=self.change_status) # Izkrāsojam vietu ar zaļu krāsu un liksim o 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 izvēlētu vietu)
```

```
    self.status = 2
```

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

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

```
    self.status = 0
```

```
    self.button.config(bg="green", text="o") # zaļš reprezentē brīvo vietu.
```

```
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="X") # Izmainām krāsu uz sarkanu un izmainām
tekstu uz pogas uz "X".
```

```
class Teatris:
```

```
    # Teātra klase.
```

```
def __init__(self, theater, rindas, kolonnas):
```

```
    self.seats = []
```

```
    self.selected_seats = [] # Izvēlētas 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 sēdvietā sākotnēji ir aizņemta, tad palieliniet
            skaitītāju.

            rinda.append(seat)

    self.seats.append(rinda)

    self.buy_button = tkinter.Button(theater, text="Nopirkt izvēlētajās 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 izvēlētajām sēdvietām izmantojot pilnu pārlasi caur katru vietu (rindu un
    kolonnu).

    selected_seats = [] # Tukšais saraksts ar izvēlētajām sēdvietām.

    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 (izvēlēta), 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. Izsauk seat.occupy() visiem sēdvietam kas ie izvēlētas un palielina num_occupied_seats par katru izvēlēto vietu (tas ir nepieciešams, 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 # Palielinām 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()) # Nolasām vērtības no rindas entry.
```

```
        kolonnas = int(kolonnas_entry.get()) # Nolasām vērtības no kolonnas (sēdvietas) entry.
```

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

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

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

```
        Teatris(zale, rindas, kolonnas) # Izsauk 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 ievādīt 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"

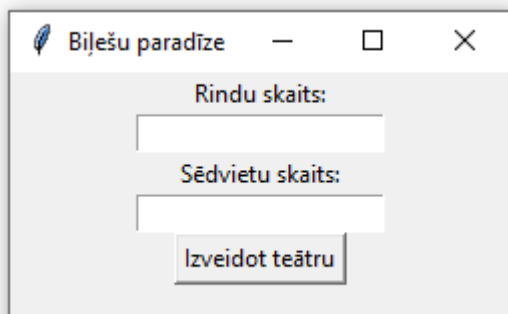
poga_izveidot.pack()

logs.mainloop() # Lai logs būtu redzāms.

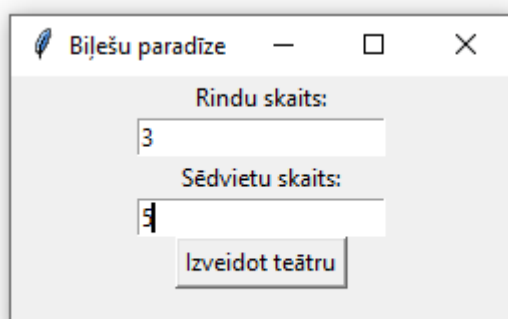
```

Testa piemēri:

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



2) Ievadam rindas un sēdvietu skaitu.



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.



X	○	X	○	X
○	X	○	X	X
○	○	X	○	X
Nopirkt izvēlētās biļetes				

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



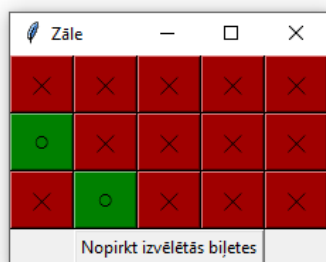
5) Uzklikšķinot uz “Nopirkt izvēlētās biļetes” izvēlētās sēdvietas tiek izmainītas uz aizņemtām.



6) Vai izvēlēties lielu skaitu vienlaikus un nopirkt.



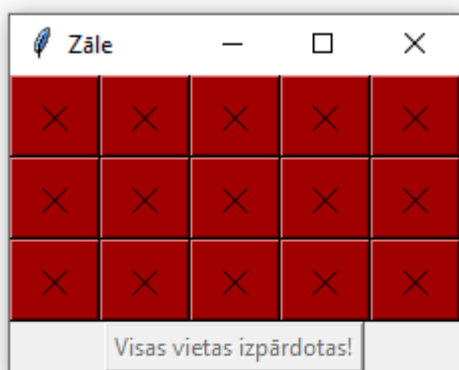
7)



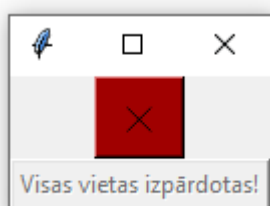
8)



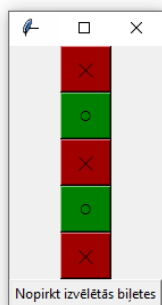
9) Var izpirkt visu un tad pogai teksts izmainīsies uz "Visas vietas izpārdotas!" un poga tiek bloķēta.



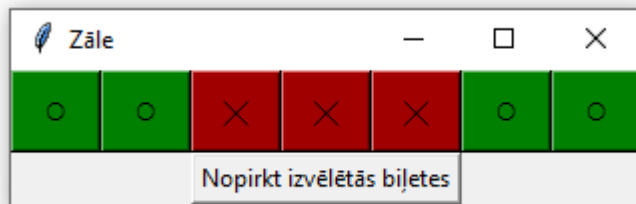
10) Ja tiek izveidots teātris 1x1, tad ja vieta ir aizņemta, tad uzreiz poga būs bloķēta un teksts būs "Visas vietas izpārdotas!".



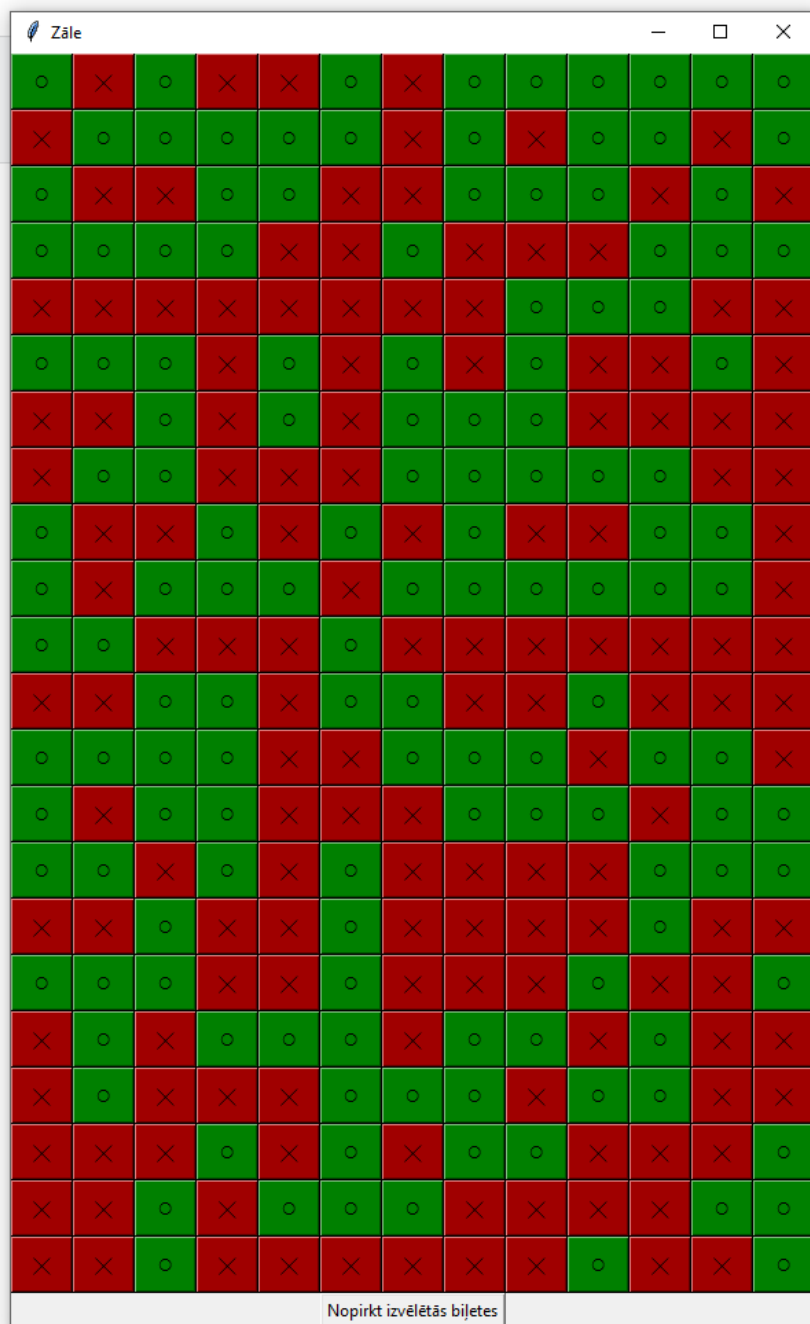
11)



12)



13)



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.

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 inicializēšana.
```

```
        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="X", width=5, height=2) #
        Izkrāsojam vietu ar sarkanu krāsu un liksim X simbolu.
```

```

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

    self.button = tkinter.Button(theater, bg="green", text="o", width=5, height=2,
command=self.change_status) # Izkrāsojam vietu ar zaļu krāsu un liksim o 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 izvēlētu vietu)

        self.status = 2

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

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

        self.status = 0

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

    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="X") # Izmainām krāsu uz sarkanu un izmainām
tekstu uz pogas uz "X".


class Teatris:

    # Teātra klase.

    def __init__(self, theater, rindas, kolonnas):

        self.seats = []

```

```

self.selected_seats = [] # Izvēlētas 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 sēdvietā sākotnēji ir aizņemta, tad palieliniet
                    skaitītāju.

                    rinda.append(seat)

            self.seats.append(rinda)

        self.buy_button = tkinter.Button(theater, text="Nopirkt izvēlētas 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 (izvēlēta), 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 nepieciešams, 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 # Palielinām 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 ievādītās rindas un kolonnu vērtības no entry, nodzēs palīglogu teātra definēšanai, un izveido "zale" windows un izsauc klasi Teatris.
```

```
        rindas = int(rindas_entry.get()) # Nolasām vērtības no rindas entry.
```

```
        kolonnas = int(kolonnas_entry.get()) # Nolasām vērtības no kolonnas (sēdvietas) entry.
```

```
        logs.destroy() # Nodzēsam logu, kur mēs prāšijam 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 ievādīt 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()
```

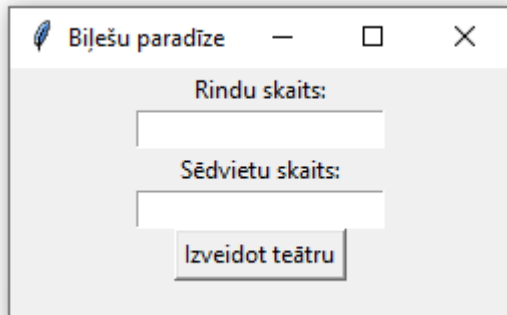
```
poga_izveidot = tkinter.Button(logs, text="Izveidot teātru",  
command=Define_Theater.izveidot_teatri) # Poga "Izveidot teātru" papildlogam. Izsauk komandi "Define_Theater.izveidot_teatri"
```

```
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.



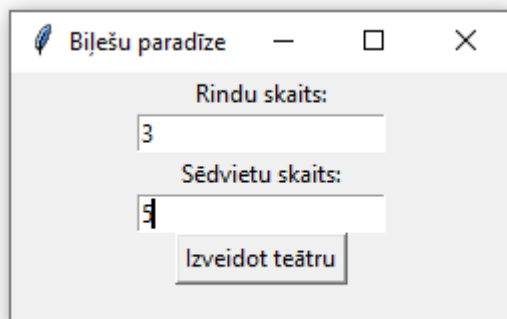
Biļešu paradīze

Rindu skaits:

Sēdvietu skaits:

Izveidot teātru

2) Ievadam rindas un sēdvietu skaitu.



Biļešu paradīze

Rindu skaits:

3

Sēdvietu skaits:

5

Izveidot teātru

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.



Zāle

X	O	X	O	X
O	X	O	X	X
O	O	X	O	X

Nopirkt izvēlētās biļetes

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



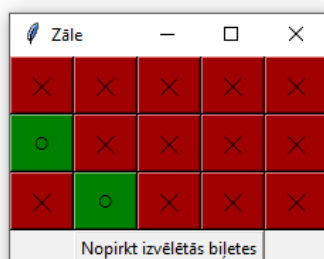
5) Uzklikšķinot uz “Nopirkt izvēlētās biļetes” izvēlētas sēdvietas tiek izmainītas uz aizņemtam.



6) Vai izvēlēties lielu skaitu vienlaikus un nopirkt.



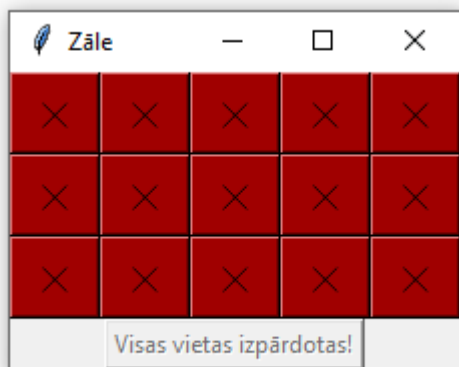
7)



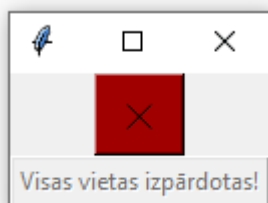
8)



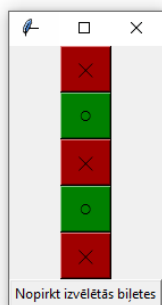
9) Var izpirkt visu un tad pogai teksts izmainīsies uz "Visas vietas izpārdotas!" un poga tiek bloķēta.



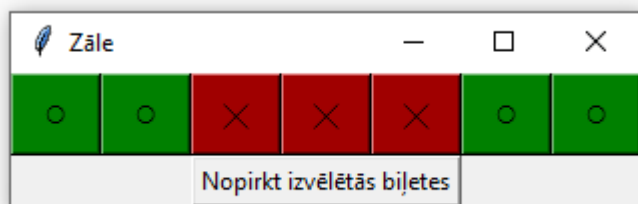
10) Ja tiek izveidots teātris 1x1, tad ja vieta ir aizņemta, tad uzreiz poga būs bloķēta un teksts būs "Visas vietas izpārdotas!".



11)



12)



13)

