**1MPR10\_Simona\_Bļinova sb24037**

**1.uzdevums**

Programma, kas realizē eksāmena simulāciju.

**Kods:**

import random

import string

studenti = set()

biletes = set()

for i in range(1, 57):

studenti.add(i)

for i in range(65, 91):

biletes.add(chr(i))

#print(studenti)

#print(biletes)

studentu\_atbildes = []

for i in range(10):

while True:

st = random.randint(1, 56)

b = random.choice(string.ascii\_uppercase)

#print(st)

#print(chr(b))

if b in biletes and st in studenti:

studenti.remove(st)

biletes.remove(b)

break

#print(studenti)

#print(biletes)

print(f'Students: {st}, biļete: {b}')

studentu\_atbildes.append((st, b))

#print(studentu\_atbildes)

while len(studenti) > 0:

#print(studentu\_atbildes)

a = studentu\_atbildes[0]

studentu\_atbildes.pop(0)

biletes.add(a[1])

while True:

st = random.randint(1, 56)

b = random.choice(string.ascii\_uppercase)

#print(st)

#print(chr(b))

if b in biletes and st in studenti:

studenti.remove(st)

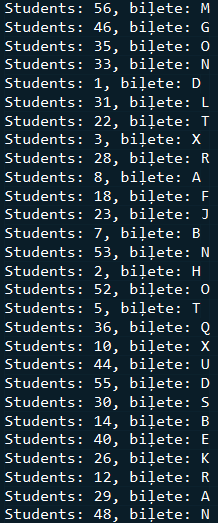
biletes.remove(b)

break

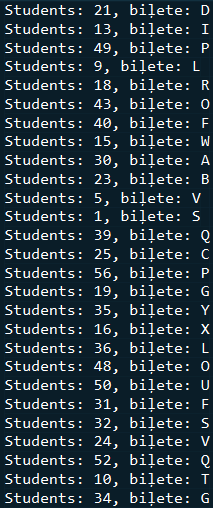
print(f'Students: {st}, biļete: {b}')

studentu\_atbildes.append((st, b))

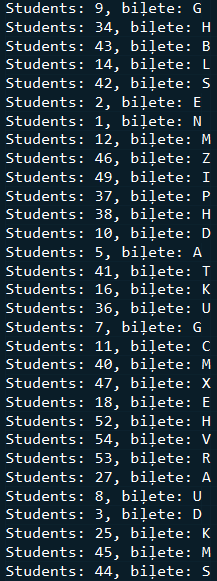
Testa piemērs(1)



Testa piemērs(2)



Testa piemērs(3)



**2.uzdevums**

Programma, kas izveido klasi Taisnsturis un izvada objekta perimetru un laukumu.

**Kods:**

class Taisnsturis():

'''

Klase 'Taisnsturis' pārstāv vienkāršu taisnstūra modeli ar pamatinformāciju:

garums, platums

Šī klase nodrošina metodes laukuma, perimetra aprēķiniem un informācijas izvadei.

Lauki:

garums (int): Taisntūra garums

platums (int): Taisnstūra platums

Metodes:

perimetrs()

laukums()

\_\_str\_\_()

'''

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

'''

Konstruktors (inicializators).

Piešķir sākotnējās vērtības laukiem.

Argumenti:

garums (int): Taisntūra garums

platums (int): Taisnstūra platums

'''

self.garums = garums

self.platums = platums

def perimetrs(self):

'''

Metode.

Aprēķina objekta perimetru.

'''

return f'Perimetrs: {(self.garums + self.platums) \* 2}'

def laukums(self):

'''

Metode.

Aprēķina objekta laukumu.

'''

return f'Laukums: {self.garums \* self.platums}'

def \_\_str\_\_(self):

'''

Metode.

Izvada informāciju par objektu.

'''

return f'Taisnstūris {self.garums} x {self.platums}'

def galvena\_programma():

x = int(input('Ievaidet taisnstūra garumu --> '))

y = int(input('Ievaidet taisnstūra platumu --> '))

z = Taisnsturis(x, y)

print(z)

print(z.perimetrs())

print(z.laukums())

if \_\_name\_\_ == '\_\_main\_\_':

galvena\_programma()

#mans\_taisnsturis = Taisnsturis(4, 4)

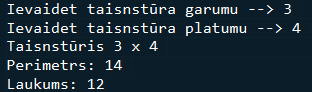
##mans\_taisnsturis2 = Taisnsturis(6, 2)

#print(mans\_taisnsturis)

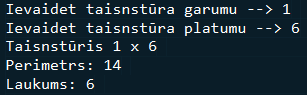
#print(mans\_taisnsturis.perimetrs())

#print(mans\_taisnsturis.laukums())

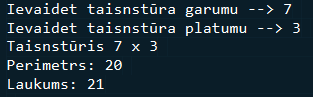
Testa piemērs(1)



Testa piemērs(2)



Testa piemērs(3)



**3.uzdevums**

Programma, kas izveido klasi Trijsturis ar noteiktam metodēm.

**Kods:**

import math

class Trijsturis:

'''

Klase 'Trijsturis' pārstāv vienkāršu trijstūra modeli ar pamatinformāciju:

mala1, mala2, mala3

Šī klase nodrošina metodes laukuma, perimetra, apvilktas un ievilktas riņķa lījas radiusa aprēķiniem un trijstūra malu garumu izvadi.

Lauki:

mala1 (int): 1. trijstūra mala

mala2 (int): 2. trijstūra mala

mala3 (int): 3. trijstūra mala

Metodes:

perimetrs()

laukums()

ievilktas\_rinka\_linijas\_radiuss()

apvilktas\_rinka\_linijas\_radiuss()

\_\_str\_\_()

'''

def \_\_init\_\_(self, mala1, mala2, mala3):

'''

Konstruktors (inicializators).

Piešķir sākotnējās vērtības laukiem.

Argumenti:

mala1 (int): 1. trijstūra mala

mala2 (int): 2. trijstūra mala

mala3 (int): 3. trijstūra mala

'''

self.mala1 = mala1

self.mala2 = mala2

self.mala3 = mala3

def perimetrs(self):

'''

Metode.

Aprēķina objekta perimetru.

'''

return self.mala1 + self.mala2 + self.mala3

def laukums(self):

'''

Metode.

Aprēķina objekta laukumu.

'''

p = self.perimetrs()

#print(p)

pusp = p / 2

return round(math.sqrt(pusp \* (pusp - self.mala1) \* (pusp - self.mala2) \* (pusp - self.mala3)), 2)

def ievilktas\_rinka\_linijas\_radiuss(self):

'''

Metode.

Aprēķina objekta ievilktas rinka linijas radiusu.

'''

s = self.laukums()

p = self.perimetrs()

return round(2 \* s / p, 2)

def apvilktas\_rinka\_linijas\_radiuss(self):

'''

Metode.

Aprēķina objekta apvilktas rinka linijas radiusu.

'''

s = self.laukums()

return round(self.mala1 \* self.mala2 \* self.mala3 / 4 / s, 2)

def \_\_str\_\_(self):

'''

Metode.

Izvada visu sākotnējo informāciju par objektu.

'''

return f'Trijstūris {self.mala1}x{self.mala2}x{self.mala3}'

def galvena\_programma():

m1 = int(input('Ievaidet trijstūra 1. malu --> '))

m2 = int(input('Ievaidet trijstūra 2. malu --> '))

m3 = int(input('Ievadiet trijstūra 3. malu --> '))

if m1+m2>m3 and m1+m3>m2 and m2+m3>m2:

trijsturis = Trijsturis(m1, m2, m3)

print(f'Perimetrs: {trijsturis.perimetrs()}')

print(f'Laukums: {trijsturis.laukums()}')

print(f'Ievilktas riņķa līnijas laukums: {trijsturis.ievilktas\_rinka\_linijas\_radiuss()}')

print(f'Apvilktas riņķa līnijas laukums: {trijsturis.apvilktas\_rinka\_linijas\_radiuss()}')

else:

print('Tadas malas trijstūrim nav iespējamas!')

exit()

if \_\_name\_\_ == '\_\_main\_\_':

galvena\_programma()

Testa piemērs(1)

A screen shot of a computer code

AI-generated content may be incorrect.

Testa piemērs(2)

A screen shot of a computer code

AI-generated content may be incorrect.

Testa piemērs(3)

A screen shot of a computer screen

AI-generated content may be incorrect.