# 31.2.3 Testavimas ir taisymas.

## Ką reikia mokėti

* Diskutuojama apie testavimo strategijas
* įprasti ir ekstremalūs duomenų rinkiniai
* Konstruojami pradiniai duomenų rinkiniai, mokomasi naudojantis šiais rinkiniais ieškoti klaidų ir jas taisyti

## Testavimo strategija

Programų inžinerijoje, **testavimo strategija** - tai planas, apibūdinantis, kaip bus testuojama programinė įranga jos kūrimo cikle. Testavimo strategijos tikslas - išsamiai aprašyti veiksmus, kurie padės pasiekti kokybės užtikrinimo tikslus. Testavimo strategija atskiriems projektams gali skirtis. Testavimo strategiją kuria ir aptaria visi projekto dalyviai, kad būtų išvengta nesusipratimų ir kad visi žinotų savo vaidmenis ir atsakomybes. Laikui bėgant strategija reguliariai peržiūrima ir atnaujinama, atsižvelgiant į organizacijos ir produkto pokyčius. Strategija padeda suvienyti skirtingus kokybės užtikrinimo srities dalyvius: suvienodina naudojamą terminologiją, nustato testavimo lygius ir planuoja išteklius.

Testavimo strategija **mokyklos lygyje** suprantama kaip apibrėžtas būdas, nusakantis kaip testuoti sukurtą programą, paprastai turinčią **įvestį** (duomenis) ir **išvestį** (rezultatus), neįtraukiant našumo, naudotojų sąsajos, naudotojų patirties ir kitų aspektų.

Kadangi mokyklos kurse neįtraukiamas automatizuotas testavimas, strategija iš esmės susiveda į duomenų rinkinių konstravimą, kuriuos naudojant bus ieškoma klaidų. Dažniausiai naudojami yra šie būdai: **ekvivalentinio sudalijimo**, **ribinių reikšmių** ir **ekstremalių reikšmių**.

Šie būdai ir bus pagrinde nagrinėjami šioje medžiagoje.

## Ekvivalentinis sudalijimas (angl. *Equivalence partitioning*)

**Ekvivalentinis sudalijimas** - tai programinės įrangos testavimo metodas, kai įvesties duomenys yra suskirstomi į grupes (vadinamas ekvivalentinėmis klasėmis) pagal tam tikrus kriterijus. Tokiu būdu sudarytos grupės leidžia pasirinkti keletą testavimo atvejų iš kiekvienos grupės ir taip išbandyti didelę dalį funkcionalumo.

Duomenų rinkinys priklauso tai pačiai *ekvivalentinei klasei*, jei kiekvienas duomenų rinkinio egzempliorius paliečia tą pačią programos kodo dalį. Idėja ta, kad vykdant tas pačias instrukcijas, daugeliu atvejų programa veiks taip pat (teisingai ar neteisingai). Todėl, norint ištestuoti ekvivalentinę klasę - reikia bent vieno duomenų rinkinio egzemplioriaus.

Pavyzdys (iš uždavinių):

if temperatūra > 10 and temperatūra < 20:
 print("Vidutinė temperatūra")
elif temperatūra >= 20:
 print("Šilta diena")
else:
 print("Šalta diena")

Šiame programos fragmente matome sąlygos sakinį, kuriame yra nusakytas intervalas (10, 20). Vadinasi galime suskirstyti duomenų rinkinius į tokias **3** klases:

 ... 8 9 10 11 ............. 19 20 21 22 .....
 --------------|-------------------|---------------------
 šalta diena vidutinė temp. šalta diena

Taigi, pagal ekvivalentinio sudalijimo idėją, testavimui būtina paimti 3 *temperatūra* kintamojo reikšmes iš kiekvienos klasės. Tarkim: **8**, **15** ir **22**.

Ar ekvivalentinei klasei reikia paimti daugiau reikšmių? Jei ekvivalentinės klasės sudarytos pagal programinį kodą, kuris toks paprastas, kaip šiame pavyzdyje, tuomet - ne. Tačiau, dažnai testavimo rinkinius mes sudarome ne iš programinio kodo, o iš programos specifikacijos ir visi programos keliai nėra žinomi. Todėl ekvivalentinei klasei testuoti paimama daugiau rinkinių.

*Šiuose uždaviniuose laikysime, kad ekvivalentinei klasei reikia parinkti bent vieną duomenų rinkinio egzempliorių, tačiau nebus klaida, nei parenkama daugiau.*

## Ribinių reikšmių analizė

**Ribinių reikšmių analizė** - tai programinės įrangos testavimo metodas, kai testavimo atvejai yra sudaromi remiantis ribinėmis reikšmėmis tam tikrame intervale. Pagrindinė šio metodo idėja yra tokia: tikėtina, kad klaidos pasirodys būtent ties ribinėmis vertėmis, nes sistemos reakcija į kraštutines reikšmes gali skirtis nuo reakcijos į vidutines reikšmes.

Prisiminkime ekvivalentinį sudalijimą. Įvesties duomenys gali būti suskirstyti į klases, kuriose kiekvienos klasės vertės sistemoje elgiasi panašiai. Tarp šių klasių atsiranda ribos, kur reikšmės keičia savo klasę - tai vadinama ribine verte.

Ribinės reikšmės turi būti parenkamos taip, kad mažiausias pasikeitimas įtakotų klasės (arba intervalo) pasikeitimą. Pavyzdžiui, jei turime *integer* tipo kintamąjį, tuomet mažiausias pasikeitimas - vienetu. Jei tai skaičius su kableliu, tuomet - tai mažiausia įmanoma reikšmė.

Panagrinėkime pavyzdį:

if temperatūra > 10 and temperatūra < 20:
 print("Vidutinė temperatūra")
elif temperatūra >= 20:
 print("Šilta diena")
else:
 print("Šalta diena")

Šiame programos fragmente matome sąlygos sakinį, kuriame yra nusakytas interavalas (10, 20). Vadinasi, klasės pasikeitimas vyksta intervalo ribom:

 ... 8 9 10 11 ............. 19 20 21 22 .....
 --------------|-------------------|---------------------
 šalta diena vidutinė temp. šalta diena

Šiuo atveju reikia testuoti su šiomis kintamojo *temperatūra* reikšmėm: **10**, **11**, **19**, **20**. Jeigu tai skaičius su kableliu, kuriam skiriami du skaitmenys po kablelio, tuomet reiktų testuoti su reikšmėm: **10.99**, **10.00**, **19.99**, **20,00**.

## Ekstremalių reikšmių analizė

Šis metodas skirtas patikrinti, kaip sistema veikia esant labai didelėms arba labai mažoms (ekstremalioms) įvesties reikšmėms, kurios yra už normalaus leistino diapazono ribų. Ekstremalioms reikšmėms taip pat kartais priskiriamos tokios reikšmės, kurios yra jautrios taikomąjai sričiai. Pavyzdžiui, skaičius **0** irgi gali būti laikoma ekstremalia reikšme ten kur naudojama dalyba ir gali pasireikšti dalybos iš nulio operacija.

Ekstremalių reikšmių testavimas padeda nustatyti sistemos elgesį neįprastose situacijose, siekiant užtikrinti, kad sistema nesugriūtų ir tinkamai praneštų apie klaidas, kai įvedamos reikšmės, kurios yra už įprasto veikimo ribų.

Kokios tos ekstremalios reikšmės - priklauso nuo taikomosios srities. Ekstremalias reikšmes siekama parinkti tokias, kurios perpildytų kintamojo tipų galimų reikšmių aibę arba masyvą.

*Kadangi nėra tikslaus mato, kokia turi būti ekstremali reikšmė, šiuose uždaviniuose laikysime, kad ekstremali reikšmė turi bent viena eile būti didesnė/mažesnė už teisingų reikšmių intervalą arba kitų testinių duomenų reikšmes.*

Panagrinėkime pavyzdį, kuris neapsaugotas nuo klaidingo veikimo:

diskriminantas = b\*\*2 - 4\*a\*c
if diskriminantas > 0:
 root1 = (-b + math.sqrt(diskriminantas)) / (2\*a)
 root2 = (-b - math.sqrt(diskriminantas)) / (2\*a)
 print(f"Šaknys: x1 = {root1}, x2 = {root2}")

Šiame programos fragmente matome kvadratinės lygties diskriminanto skaičiavimą. Kintamojo *a* ekstremalių reikšmių pavyzdys būtų: **-1000000**, **-0.0000001**, **0**, **0.0000001**, **1000000**. Kaip matote, tarp ekstremalių reikšmių įtraukėme ir tas, kurios artimos **0**. Šios reikšmės leistų aptikti, kad kintamasis *a* negali būti lygus nuliui ir dėl to reiktų pridėti atitinkamą patikrinimą.

## Testavimo duomenys ir testavimo atvejai

**Testavimo duomenys** - tai specifiniai duomenys (vertės ar įvestys), naudojami programinės įrangos veikimui patikrinti. Testavimo duomenys gali būti skirtingų tipų, priklausomai nuo programos, pvz., skaičiai, tekstai, failai. Kai turime **tik** testavimo duomenis, programos veikimą turime tikrinti patys.

**Testavimo atvejai** - tai testavimo duomenys kartu su tikėtinais rezultatais, kurie turėtų būti gaunami įvykdžius programą su testavimo duomenimis. Testavimo atvejai leidžia automatizuoti programos patikrinimą.

## Testavimo metodologijos pagal žinias apie sistemos struktūrą

### Juodosios dėžės testavimas

Testavimo metodas, kai testuotojas neturi informacijos apie sistemos vidinį veikimą ir analizuoja tik jos išorinius veikimo aspektus. Tikrinama, ar sistema tinkamai reaguoja į įvairias įvestis ir ar pateikia teisingus rezultatus. Šis metodas dažnai naudojamas funkciniam testavimui. Tinka visos aptartos strategijos: ribinių ir ekstremalių reikšmių analizė, ekvivalentinio sudalijimo metodas.

### Baltosios dėžės testavimas

Testavimo metodas, kai testuotojas turi prieigą prie sistemos vidinio kodo ir struktūros. Tikrinama, kaip kiekviena kodo dalis veikia, ir ieškoma klaidų kodo lygmenyje. Šis metodas dažnai naudojamas vienetų testavimui. Atlikus kodo analizę gali būti taikomos aptartos strategijos.

### Pilkosios dėžės testavimas

Tai mišrus testavimo metodas, kai testuotojas turi ribotą prieigą prie sistemos vidinio veikimo (pvz., pagrindinių struktūrų ar duomenų bazės). Naudojamas funkciniams testams atlikti kartu su pagrindinėmis techninėmis žiniomis apie sistemą.

## Testavimo lygiai

Testavimo lygiai yra skirtingi programinės įrangos testavimo būdai, atliekami skirtinguose programos kūrimo žingsniuose, siekiant užtikrinti kokybę kiekviename etape. Kiekvienas lygis skirtas tam tikram tikslui ir padeda testuotojams patikrinti skirtingus sistemos aspektus.

## Vienetų testavimas (angl. *Unit testing*)

Testavimo lygis, kuriame atskiri programos komponentai arba „vienetai“ (pvz., funkcijos, metodai ar moduliai) yra testuojami atskirai. Kai kurdami programą iškviečiame vieną funkciją su tam tikrais parametrais ir pasitikrinam jos rezultatus - tai vienetų testavimas. Paprastai jis atliekamas automatizuotai.

## Integracinis testavimas (angl. *Integration testing*)

Testavimo lygis, kuriame testuojama, kaip skirtingi programinės įrangos komponentai veikia kartu. Tai padeda nustatyti, ar nėra klaidų komponentų sąveikoje. Kai mes tikriname programą, kuri nuskaito duomenis iš failo ir atlieka skaičiavimus - tai jau integracinis testavimas.

## Sistemos testavimas (angl. *System testing*)

Testavimo lygis, kuriame mes tikriname visos programos veikimą, įtraukiant ir naudotojo sąsają. Šie testai gali būti automatizuoti, tačiau dažnai atliekami rankiniu būdu, nes kai kuriose taikomosiose srityse automatizuoti visos sistemos patikrinimą yra sudėtinga.

## Programos funkcijų patikra

Čia kalbama apie patikrą, kuri atliekama prieš priduodant (arba - priduodant) progamą.

### Funkcinis testavimas

**Funkcinis testavimas** yra skirtas patikrinti ar programa teisingai veikia pagal iškeltus funkcinius reikalavimus. Į funkcinį testavimą įeina vienetų, integraciniai, sistemos testai ir kiti testavimo būdai, kurie tikrina sistemos veikimą pagal jos specifikaciją.

### Priėmimo testavimas

**Priėmimo testavimas** yra skirtas patikrinti ar sukurta teisinga programa. Ar programa daro tai ko reikia naudotojui. Gali kilti klausimas - jei sistema korektiškai veikia pagal funkcinius reikalavimus, ar ji gali būti neteisinga. Taip, vienas iš variantų - klaidos padarytos specifikuojant (aprašant) būsimą programą. Priėmimo testavimas dažniausiai atliekamas rankiniu būdu, kartu su programos savininku.

## Kitos testavimo metodologijos

### Regresinis testavimas

**Regresinis testavimas** - tai testavimas, kuris atliekamas po to, kai programinė įranga buvo patobulinta, pakeista arba ištaisyta. Pagrindinis regresinio testavimo tikslas yra patikrinti, ar nauji pakeitimai nesukėlė neigiamų pasekmių ir nesugadino anksčiau veikusių sistemos dalių. Todėl testavimo duomenys yra svarbūs ne tik programos kūrimo metu.

### Našumo testavimas

**Našumo testavimas** - tai testavimas, skirtas įvertinti, kaip sistema veikia esant skirtingoms apkrovoms, ir nustatyti jos galimybes, kai naudojama realiomis sąlygomis. Tai yra, kai ją naudoja daug vartotojų, kai atliekamos sudėtingos užklausos arba atliekami sudėtingi skaičiavimai. Našumas apima net tik programos geritį, bet ir reikalavimus atminčiai, tinklui ir kt. Su našumo testavimu susijęs **stresinis testavimas**, kurio tikslas - nustatyti programos našumo ribas ir atsparumą.

### Scenarijų testavimas

**Scenarijų testavimas** - tai testavimo metodas, kurio metu kuriami realistiniai scenarijai (veiksmų sekos), atspindintys, kaip vartotojas naudotų programinę įrangą. Šio metodo tikslas - patikrinti, kaip sistema veikia imituojant darbo su programa procesus. Dažnai naudojamas priėmimo testavimo metu.

## Testais paremtas programų kūrimas (angl. *Test driven development*)

Tai programavimo metodas, kai programos kūrimas prasideda nuo testų rašymo (testinių duomenų rinkinių kūrimo), o tik tada pereinama prie funkcionalumo programavimo. Pagrindinis tikslas - užtikrinti, kad kiekviena parašyta kodo dalis atitinka reikalavimus ir veikia teisingai.

Tai įdomus programos kūrimo būdas, kurį galima išmėginti su moksleiviais. Kaip pavyzdys, galėtų būti tokie būtų programos kūrimo žingsniai:

* Sukurkite duomenų failą
* Įgyvendinkite duomenų nuskaitymą
* Sukurkite minimalią skaičiavimų funkciją, kad pradiniai testai (duomenys) praeitų
* Patobulinkite kodą (kodas optimizuojamas, kad būtų kuo paprastesnis, efektyvesnis ir patikimesnis).
* Iteruokite (pridėkite daugiau testinių duomenų, kurie aptiktų paliktas klaidas; tobulinkite programą, kad testai praeitų).

## Kodo, šakų padengimas

Kodo padengimas (angl. *Code coverage*) - tai metrika, kuri matuoja, kokia dalis programinio kodo buvo patikrinta testavimo metu. Kodo padengimo analizė padeda nustatyti, kurios kodo dalys buvo išbandytos, ir nustatyti tas, kurios dar neturi testų.

Kodo padengimo tipai:

* Eilučių padengimas (angl. *Line coverage*): matuoja, kiek kodo eilučių buvo vykdyta testavimo metu. Jei kodo eilutė nebuvo vykdyta nė su vienais testiniais duomenimis, ji laikoma nepadengta.
* Šakų padengimas (angl. *Branch coverage*): matuoja, ar visos loginės šakos (pvz., sąlygos, kaip *if*, *else*, *elif/else if*, *switch*) buvo vykdytos. Kitaip sakant, tikrinama, ar įvykdomas kiekvienas sąlygos blokas.
* Sąlygų padengimas (angl. *Condition coverage*): kiekviena loginė sąlyga tikrinama atskirai, siekiant užtikrinti, kad buvo patikrinti visi sąlygos rezultatai (pvz., tikrinama tiek true, tiek false rezultatai). Sąlygų padengimas paprastai įtraukia daugiau testų nei šakų padengimas, nes *if* sakinys gali įtraukti kelias sąlygas.

*Testuose įtraukti šakų padengimo ir sąlygų padengimo uždaviniai.*

# 31.2.3 Testavimas ir taisymas.

## Ką reikia mokėti:

* Diskutuojama apie testavimo strategijas
* įprasti ir ekstremalūs duomenų rinkiniai
* Konstruojami pradiniai duomenų rinkiniai, mokomasi naudojantis šiais rinkiniais ieškoti klaidų ir jas taisyti

## Praktinės užduotys

### 1.1 užduotis

Programuotojas sukūrė programą kvadratinės lygties šaknų skaičiavimui. Taip pat parašė nedidelę funkciją skaičiavimų testavimui (testuok()). Dabar reikia paruošti testavimo duomenų failą **testai.csv**, kurio kiekvienoje eilutėje būtų lygties koeficientai. Parenkite tokį testai.csv failą, kad būtų patikrintos visos funkcijos skaiciuok() šakos.

C++ programa:

#include <iostream>
#include <fstream>
#include <sstream>
#include <cmath>
using namespace std;

void skaiciuok(double a, double b, double c) {
 double diskriminantas = b\*b - 4\*a\*c;

 if (a == 0) {
 cout << "a negali būti lygus 0. Tai nėra kvadratinė lygtis." << endl;
 } else if (diskriminantas > 0) {
 double root1 = (-b + sqrt(diskriminantas)) / (2\*a);
 double root2 = (-b - sqrt(diskriminantas)) / (2\*a);
 cout << "Šaknys: x1 = " << root1 << ", x2 = " << root2 << endl;
 } else if (diskriminantas == 0) {
 double root = -b / (2\*a);
 cout << "Viena reali šaknis: x = " << root << endl;
 } else {
 cout << "Lygtis neturi realių šaknų." << endl;
 }
}

void testuok() {
 ifstream file("testai.csv");
 string line;

 while (getline(file, line)) {
 stringstream ss(line);
 string value;
 double a, b, c;

 getline(ss, value, ',');
 a = stod(value);
 getline(ss, value, ',');
 b = stod(value);
 getline(ss, value, ',');
 c = stod(value);

 cout << "Koeficientai: a = " << a << ", b = " << b << ", c = " << c << endl;
 skaiciuok(a, b, c);
 }
}

int main() {
 testuok();
 return 0;
}

Python programa:

import math

def skaiciuok(a, b, c):
 diskriminantas = b\*\*2 - 4\*a\*c
 if a == 0:
 print("a negali būti lygus 0. Tai nėra kvadratinė lygtis.")
 elif diskriminantas > 0:
 root1 = (-b + math.sqrt(diskriminantas)) / (2\*a)
 root2 = (-b - math.sqrt(diskriminantas)) / (2\*a)
 print(f"Šaknys: x1 = {root1}, x2 = {root2}")
 elif diskriminantas == 0:
 root = -b / (2\*a)
 print(f"Viena reali šaknis: x = {root}")
 else:
 print("Lygtis neturi realių šaknų.")

def testuok():
 with open('testai.csv', 'r') as file:
 lines = file.readlines()
 for line in lines:
 line = line.strip()
 if line:
 values = line.split(',')
 a = int(values[0])
 b = int(values[1])
 c = int(values[2])
 print(f"Koeficientai: a = {a}, b = {b}, c = {c}")
 skaiciuok(a, b, c)

testuok()

#### Vertinimas

| Nr. | Kas vertinama | Balai |
| --- | --- | --- |
| 1. | Yra keoficientų rinkinys a == 0 šakos patikrinimui | 1,5 |
| 2. | Yra keoficientų rinkinys diskriminantas > 0 šakos patikrinimui | 1,5 |
| 3. | Yra keoficientų rinkinys diskriminantas == 0 šakos patikrinimui | 1,5 |
| 4. | Yra keoficientų rinkinys diskriminantas < 0 šakos patikrinimui | 1,5 |
|  | Viso: | 6 |

#### Teisingas atsakymas

Minimalus rinkinys visų šakų patikrinimui:

0,2,1
1,6,1
2,4,2
16,2,6

#### Taškų skaičius

6

#### Sudėtingumo lygis

3

#### Ugdomos kompetencijos

Pažinimo

### 1.2 užduotis

Programuotojas sukūrė programą kvadratinės lygties šaknų skaičiavimui, kuri dar neapsaugota nuo neteisingo veikimo. Taip pat parašė nedidelę funkciją skaičiavimų testavimui (testuok()). Minimalų testų rinkinį surašė į failą **testai.csv**, kurio kiekvienoje eilutėje yra lygties koeficientai. Patobulinkite skaiciuok() funkcijos kodą, kad būtų aptinkamos įvairios kvadratinės lygties šaknų skaičiavimo išlygos (kai lygtis nėra kvadratinė, kai lygtis turi 2 šaknis, kai lygtis turi 1 šaknį ir kai lygtis realių šaknų neturi). Duotas testavimo failas leis jums pasitikrinti savo sprendimą.

**testai.csv** failas:

0,2,1
1,6,1
2,4,2
16,2,6

C++ programa:

#include <iostream>
#include <fstream>
#include <sstream>
#include <cmath>
using namespace std;

void skaiciuok(double a, double b, double c) {
 double diskriminantas = b\*b - 4\*a\*c;

 if (diskriminantas > 0) {
 double root1 = (-b + sqrt(diskriminantas)) / (2\*a);
 double root2 = (-b - sqrt(diskriminantas)) / (2\*a);
 cout << "Šaknys: x1 = " << root1 << ", x2 = " << root2 << endl;
 }
}

void testuok() {
 ifstream file("testai.csv");
 string line;

 while (getline(file, line)) {
 stringstream ss(line);
 string value;
 double a, b, c;

 getline(ss, value, ',');
 a = stod(value);
 getline(ss, value, ',');
 b = stod(value);
 getline(ss, value, ',');
 c = stod(value);

 cout << "Koeficientai: a = " << a << ", b = " << b << ", c = " << c << endl;
 skaiciuok(a, b, c);
 }
}

int main() {
 testuok();
 return 0;
}

Python programa:

import math

def skaiciuok(a, b, c):
 diskriminantas = b\*\*2 - 4\*a\*c
 if diskriminantas > 0:
 root1 = (-b + math.sqrt(diskriminantas)) / (2\*a)
 root2 = (-b - math.sqrt(diskriminantas)) / (2\*a)
 print(f"Šaknys: x1 = {root1}, x2 = {root2}")

def testuok():
 with open('testai.csv', 'r') as file:
 lines = file.readlines()
 for line in lines:
 line = line.strip()
 if line:
 values = line.split(',')
 a = int(values[0])
 b = int(values[1])
 c = int(values[2])
 print(f"Koeficientai: a = {a}, b = {b}, c = {c}")
 skaiciuok(a, b, c)

testuok()

#### Vertinimas

| Nr. | Kas vertinama | Balai |
| --- | --- | --- |
| 1. | Realizuota a == 0 sąlyga | 2 |
| 2. | Realizuota diskriminantas > 0 sąlyga | 2 |
| 3. | Realizuota diskriminantas == 0 sąlyga | 2 |
| 4. | Realizuota diskriminantas < 0 sąlyga | 2 |
|  | Viso: | 8 |

#### Teisingas atsakymas

Minimalus rinkinys visų šakų patikrinimui:

0,2,1
1,6,1
2,4,2
16,2,6

#### Taškų skaičius

8

#### Sudėtingumo lygis

3

#### Ugdomos kompetencijos

Pažinimo

### 2 užduotis

Programuotojas sukūrė programą, skirtą apskaičiuoti trikampio plotą pagal Herono formulę. Taip pat parašė nedidelę funkciją skaičiavimų testavimui (testuok()). Dabar reikia paruošti testavimo duomenų failą **trikampiai.csv**, kurio kiekvienoje eilutėje būtų trijų trikampio kraštinių ilgiai. Parenkite tokį **trikampiai.csv** failą, kad būtų patikrintos visos funkcijos trikampio sąlygos (ar kraštinės gali sudaryti trikampį).

C++ programa:

#include <iostream>
#include <fstream>
#include <sstream>
#include <cmath>

using namespace std;

void skaiciuok(double a, double b, double c) {
 if (a + b > c && a + c > b && b + c > a) {
 double s = (a + b + c) / 2;
 double plotas = sqrt(s \* (s - a) \* (s - b) \* (s - c));
 cout << "Trikampio plotas: " << plotas << endl;
 } else {
 cout << "Šios kraštinės negali sudaryti trikampio." << endl;
 }
}

void testuok() {
 ifstream file("trikampiai.csv");
 string line;

 while (getline(file, line)) {
 stringstream ss(line);
 string value;
 double a, b, c;
 getline(ss, value, ',');
 a = stod(value);
 getline(ss, value, ',');
 b = stod(value);
 getline(ss, value, ',');
 c = stod(value);
 cout << "Kraštinės: a = " << a << ", b = " << b << ", c = " << c << endl;
 skaiciuok(a, b, c);
 }
}

int main() {
 testuok();
 return 0;
}

Python programa:

import math

def skaiciuok(a, b, c):
 if a + b > c and a + c > b and b + c > a:
 s = (a + b + c) / 2
 plotas = math.sqrt(s \* (s - a) \* (s - b) \* (s - c))
 print(f"Trikampio plotas: {plotas}")
 else:
 print("Šios kraštinės negali sudaryti trikampio.")

def testuok():
 with open('trikampiai.csv', 'r') as file:
 lines = file.readlines()
 for line in lines:
 line = line.strip()
 if line:
 values = line.split(',')
 a = float(values[0])
 b = float(values[1])
 c = float(values[2])
 print(f"Kraštinės: a = {a}, b = {b}, c = {c}")
 skaiciuok(a, b, c)

testuok()

#### Vertinimas

| Nr. | Kas vertinama | Balai |
| --- | --- | --- |
| 1. | Yra keoficientų rinkinys a + b > c sąlygai | 1,5 |
| 2. | Yra keoficientų rinkinys a + c > b sąlygai | 1,5 |
| 3. | Yra keoficientų rinkinys b + c > a sąlygai | 1,5 |
| 4. | Yra keoficientų rinkinys, su kuriuo visos sąlygos tenkinamos | 1,5 |
|  | Viso: | 6 |

#### Teisingas atsakymas

Minimalus rinkinys visų sąlygų patikrinimui:

3,4,5
1,2,3
1,5,2
5,2,1

#### Taškų skaičius

6

#### Sudėtingumo lygis

3

#### Ugdomos kompetencijos

Pažinimo

### 3 užduotis

Programuotojas sukūrė programą, skirtą stačiakampio plotui apskaičiuoti pagal jo kraštinių ilgį. Dabar reikia paruošti testavimo duomenų failą **stačiakampiai.csv**, kuriame be teisingų duomenų būtų ribinės reikšmės, kad būtų galima patikrinti, ar programa tinkamai veikia esant netinkamoms reikšmėms. Paruoškite testus ir patobulinkite stačiakampio ploto skaičiavimo funkciją, kad ji įvertintų galimas netinkamas reikšmes.

C++ programa:

#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
using namespace std;

void skaiciuok\_plota(double ilgis, double plotis) {
 double plotas = ilgis \* plotis;
 cout << "Stačiakampio plotas: " << plotas << endl;
}

void testuok() {
 ifstream file("stačiakampiai.csv");
 string line;

 while (getline(file, line)) {
 if (!line.empty()) {
 stringstream ss(line);
 string value;
 double ilgis, plotis;
 getline(ss, value, ',');
 ilgis = stod(value);
 getline(ss, value, ',');
 plotis = stod(value);
 cout << "Ilgis: " << ilgis << ", Plotis: " << plotis << endl;
 skaiciuok\_plota(ilgis, plotis);
 }
 }
}

int main() {
 testuok();
 return 0;
}

Python programa:

def skaiciuok\_plota(ilgis, plotis):
 plotas = ilgis \* plotis
 print(f"Stačiakampio plotas: {plotas}")

def testuok():
 with open('stačiakampiai.csv', 'r') as file:
 lines = file.readlines()
 for line in lines:
 line = line.strip()
 if line:
 values = line.split(',')
 ilgis = float(values[0])
 plotis = float(values[1])
 print(f"Ilgis: {ilgis}, Plotis: {plotis}")
 skaiciuok\_plotas(ilgis, plotis)

testuok()

#### Vertinimas

| Nr. | Kas vertinama | Balai |
| --- | --- | --- |
| 1. | Patikrinamos reikšmės su 0 | 2 |
| 2. | Patikrinamos reikšmės su neigiamais skaičiais | 2 |
| 3. | Patikrinamos reikšmės su teigiamais skaičiais (teisingos reikšmės) | 2 |
| 3. | Tinkamai pakoreguota skaiciuok\_plota() funkcija | 2 |
|  | Viso: | 8 |

#### Teisingas atsakymas

Minimalus variantas:

0,5
5,0
-1,5
5,-1

Idealus variantas:

0,5 # Netinkamas ilgis, plotis teigiamas
5,0 # Ilgis teigiamas, netinkamas plotis
0,0 # Ilgis netinkamas, netinkamas plotis
-1,4 # Netinkamas neigiamas ilgis
4,-1 # Netinkamas neigiamas plotis
-1,-1 # Netinkamas neigiamas ilgis ir plotis
0.0001,5 # Labai maža reikšmė, artima nuliui, bet teigiama
5,0.0001 # Labai maža reikšmė, artima nuliui, bet teigiama
1000000,2 # Labai didelė reikšmė (didelis ilgis)
5,1000000 # Labai didelė reikšmė (didelis plotis)

Pakoreguota funkcija: C++:

void skaiciuok\_plota(double ilgis, double plotis) {
 if (ilgis <= 0 || plotis <= 0) {
 cout << "Ilgis ir plotis turi būti teigiamos reikšmės." << endl;
 } else {
 double plotas = ilgis \* plotis;
 cout << "Stačiakampio plotas: " << plotas << endl;
 }
}

Python:

def skaiciuok\_plota(ilgis, plotis):
 if ilgis <= 0 or plotis <= 0:
 print("Ilgis ir plotis turi būti teigiamos reikšmės.")
 else:
 plotas = ilgis \* plotis
 print(f"Stačiakampio plotas: {plotas}")

#### Taškų skaičius

8

#### Sudėtingumo lygis

4

#### Ugdomos kompetencijos

Pažinimo

### 4 užduotis

Programuotojas parašė programą, kuri pagal vartotojo duomenis (amžių, pajamas ir kredito balą) nustato, ar vartotojas gali gauti banko paskolą. Jums reikia parengti testavimo duomenų rinkinį. Šį kartą (programos supaprastinimui) duomenys parengiami programos kode. Parenkite tokį testavimo duomenų rinkinį, kuris vertintų ekvivalentinio sudalinimo ir ribinių reikšmių principus.

C++ programa:

#include <iostream>
#include <string>
using namespace std;

struct Gyventojas {
 int amzius;
 double pajamos;
 int kredito\_balas;
 string vardas;
};

string vertinti\_paskola(Gyventojas gyventojas) {
 if (gyventojas.amzius < 18) {
 return gyventojas.vardas + " neatitinka amžiaus kriterijaus.";
 } else if (gyventojas.amzius > 65) {
 return gyventojas.vardas + " per didelis amžius.";
 } else if (gyventojas.pajamos < 1000) {
 return gyventojas.vardas + " pajamos per mažos.";
 } else if (gyventojas.kredito\_balas < 600) {
 return gyventojas.vardas + " kredito balas per žemas.";
 } else {
 return gyventojas.vardas + " paskola gali būti suteikta.";
 }
}

int main() {
 Gyventojas gyventojai[] = {
 {20, 2200, 700, "Jonas"}
 // čia turite papildyti testavimo duomenis
 };

 int n = sizeof(gyventojai) / sizeof(gyventojai[0]);
 for (int i = 0; i < n; i++) {
 cout << vertinti\_paskola(gyventojai[i]) << endl;
 }

 return 0;
}

Python programa:

def vertinti\_paskola(gyventojas):

 if gyventojas["amzius"] < 18:

 return gyventojas["vardas"] + " neatitinka amžiaus kriterijaus."

 elif gyventojas["amzius"] > 65:

 return gyventojas["vardas"] + " per didelis amžius."

 elif gyventojas["pajamos"] < 1000:

 return gyventojas["vardas"] + " pajamos per mažos."

 elif gyventojas["kredito\_balas"] < 600:

 return gyventojas["vardas"] + " kredito balas per žemas."

 else:

 return gyventojas["vardas"] + " paskola gali būti suteikta."

# Sukuriame gyventojų sąrašą

gyventojai = [

 {"vardas": "Jonas", "amzius": 20, "pajamos": 2200, "kredito\_balas": 700},

 {"vardas": "Petras", "amzius": 17, "pajamos": 1500, "kredito\_balas": 620},

 {"vardas": "Ona", "amzius": 70, "pajamos": 3000, "kredito\_balas": 750},

 {"vardas": "Aistė", "amzius": 25, "pajamos": 800, "kredito\_balas": 580}

]

# Vertiname kiekvieno gyventojo paskolos galimybę

for gyventojas in gyventojai:

 print(vertinti\_paskola(gyventojas))

Python programa (joje naudojamos **klasės**):

class Gyventojas:
 def \_\_init\_\_(self, vardas, amzius, pajamos, kredito\_balas):
 self.vardas = vardas
 self.amzius = amzius
 self.pajamos = pajamos
 self.kredito\_balas = kredito\_balas

def vertinti\_paskola(gyventojas):
 if gyventojas.amzius < 18:
 return gyventojas.vardas + " neatitinka amžiaus kriterijaus."
 elif gyventojas.amzius > 65:
 return gyventojas.vardas + " per didelis amžius."
 elif gyventojas.pajamos < 1000:
 return gyventojas.vardas + " pajamos per mažos."
 elif gyventojas.kredito\_balas < 600:
 return gyventojas.vardas + " kredito balas per žemas."
 else:
 return gyventojas.vardas + " paskola gali būti suteikta."

# Sukuriame gyventojų sąrašą
gyventojai = [
 Gyventojas("Jonas", 20, 2200, 700)
 # Čia turite papildyti testavimo duomenis
]

# Vertiname kiekvieno gyventojo paskolos galimybę
for g in gyventojai:
 print(vertinti\_paskola(g))

#### Vertinimas

| Nr. | Kas vertinama | Balai |
| --- | --- | --- |
| 1. | Yra testavimo duomenų rinkinys, tai paskola suteikiama | 1 |
| 2. | Yra testavimo duomenų rinkiniai kiekvieno kriterijaus vidinėms reikšmėms (ekvivalentinės klasės) | 3 |
| 3. | Yra testavimo duomenų rinkiniai kiekvieno kriterijaus ribinėms reikšmėms (ribinės reikšmės) | 3 |
| 4. | Pateikta programa su testinių duomenų rinkiniu veikia teisingai | 1 |
|  | Viso: | 8 |

#### Teisingas atsakymas

Bendrinė informacija:

* Amžius:
	+ Ekvivalentinės klasės:
		- Amžius < 18 (netinkamas)
		- 18 <= Amžius <= 65 (tinkamas)
		- Amžius > 65 (netinkamas)
	+ Ribinės reikšmės:
		- 17 → „Neatitinka amžiaus kriterijaus.“
		- 18 → „Paskola gali būti suteikta.“
		- 65 → „Paskola gali būti suteikta.“
		- 66 → „Per didelis amžius.“
* Pajamos:
	+ Ekvivalentinės klasės:
		- Pajamos < 1000 (netinkamos)
		- Pajamos >= 1000 (tinkamos)
	+ Ribinės reikšmės:
		- 999 → „Pajamos per mažos.“
		- 1000 → „Paskola gali būti suteikta.“
* Kredito balas:
	+ Ekvivalentinės klasės:
		- Kredito balas < 600 (netinkamas)
		- Kredito balas >= 600 (tinkamas)
	+ Ribinės reikšmės:
		- 599 → „Kredito balas per žemas.“
		- 600 → „Paskola gali būti suteikta.“

C++ sprendimas:

// ekvivalentinės klasės
{20, 2200, 700, "Jonas"} // Tinkamas variantas
{30, 950, 650, "Ona"}, // Netinkamos pajamos
{25, 1200, 590, "Petras"}, // Žemas kredito balas
{15, 1500, 620, "Eglė"}, // Per mažas amžius
{70, 2000, 700, "Antanas"} // Per didelis amžius
// ribinės reikšmės
{17, 2200, 700, "Marija"} // Neatitinka amžiaus
{18, 2200, 700, "Nijolė"} // Atitinka amžių
{66, 2200, 700, "Saulius"} // Neatitinka amžiaus
{65, 2200, 700, "Darius"} // Atitinka amžių
{30, 999, 650, "Gerda"}, // Netinkamos pajamos
{30, 1000, 650, "Dana"}, // Tinkamos pajamos
{25, 1200, 599, "Simas"}, // Netinkamas kreditas
{25, 1200, 600, "Sonata"}, // Tinkamas kreditas

Python sprendimas:

 Gyventojas("Jonas", 20, 2200, 700), # Tinkamas variantas
 Gyventojas("Ona", 30, 950, 650), # Netinkamos pajamos
 Gyventojas("Petras", 25, 1200, 590), # Žemas kredito balas
 Gyventojas("Eglė", 15, 1500, 620), # Per mažas amžius
 Gyventojas("Antanas", 70, 2000, 700), # Per didelis amžius
 # Ribinės reikšmės
 Gyventojas("Marija", 17, 2200, 700), # Neatitinka amžiaus
 Gyventojas("Nijolė", 18, 2200, 700), # Atitinka amžių
 Gyventojas("Saulius", 66, 2200, 700), # Neatitinka amžiaus
 Gyventojas("Darius", 65, 2200, 700), # Atitinka amžių
 Gyventojas("Gerda", 30, 999, 650), # Netinkamos pajamos
 Gyventojas("Dana", 30, 1000, 650), # Tinkamos pajamos
 Gyventojas("Simas", 25, 1200, 599), # Netinkamas kredito balas
 Gyventojas("Sonata", 25, 1200, 600) # Tinkamas kredito balas

#### Taškų skaičius

8

#### Sudėtingumo lygis

4

#### Ugdomos kompetencijos

Pažinimo

### 5 užduotis

Duota tokia programos sukūrimo užduotis (reikalavimų specifikacija): Parašyti programą, kuri vertintų paskolos išdavimo galimybę gyventojui. Turi būti sukurta klasė (arba struktūra) Gyventojas, kuri turi turėti šiuos laukus:

* vardas - gyventojo vardas (teksto eilutė).
* amzius - gyventojo amžius (sveikasis skaičius).
* pajamos - gyventojo pajamos eurais (slankusis skaičius).
* kredito\_balas - gyventojo kredito balas (sveikasis skaičius).

Turi būti sukurta funkcija vertinti\_paskola(gyventojas), kuri turi atlikti šiuos vertinimus, remdamasi gyventojo duomenimis:

* Amžiaus kriterijus:
	+ Jei gyventojo amžius yra mažesnis nei 18 metai, paskola nėra suteikiama ir išvedamas pranešimas: **[vardas] - neatitinka amžiaus kriterijaus**.
	+ Jei gyventojo amžius yra didesnis nei 65 metai, paskola nėra suteikiama ir išvedamas pranešimas: **[vardas] per didelis amžius**.
* Pajamų kriterijus:
	+ Jei gyventojo mėnesinės pajamos yra mažesnės nei 1000 eurų, paskola nėra suteikiama ir išvedamas pranešimas: **[vardas] - pajamos per mažos**.
* Kredito balo kriterijus:
	+ Jei gyventojo kredito balas yra mažesnis nei 600, paskola nėra suteikiama ir išvedamas pranešimas: **[vardas] - kredito balas per žemas**.

Paskola suteikiama, jei visos sąlygos yra įvykdytos. Turi būti išvedamas pranešimas: **[vardas] - paskola suteiktina**.

Vertinami gyventojai pateikiami klasės Gyventojas sąrašu(masyvu, kolekcija). Sukūrus programą, joje parengti duomenis testavimui.

Pradinė C++ programa:

#include <iostream>
#include <string>
using namespace std;

struct Gyventojas {
// čia surašyti klasės laukus
};

string vertinti\_paskola(Gyventojas gyventojas) {
// čia įrašyti paskolos vertinimo kodą
}

int main() {
 Gyventojas gyventojai[] = {
 {20, 2200, 700, "Jonas Jonauskas"}
 // čia turite papildyti testavimo duomenis
 };

 int n = sizeof(gyventojai) / sizeof(gyventojai[0]);
 for (int i = 0; i < n; i++) {
 cout << vertinti\_paskola(gyventojai[i]) << endl;
 }

 return 0;
}

Python programa:

def vertinti\_paskola(gyventojas):

 if gyventojas["amzius"] < 18:

 return gyventojas["vardas"] + " neatitinka amžiaus kriterijaus."

 elif gyventojas["amzius"] > 65:

 return gyventojas["vardas"] + " per didelis amžius."

 elif gyventojas["pajamos"] < 1000:

 return gyventojas["vardas"] + " pajamos per mažos."

 elif gyventojas["kredito\_balas"] < 600:

 return gyventojas["vardas"] + " kredito balas per žemas."

 else:

 return gyventojas["vardas"] + " paskola gali būti suteikta."

# Sukuriame gyventojų sąrašą

gyventojai = [

 {"vardas": "Jonas Jonauskas", "amzius": 20, "pajamos": 2200, "kredito\_balas": 700},

 {"vardas": "Petras Petraitis", "amzius": 17, "pajamos": 500, "kredito\_balas": 580},

 {"vardas": "Ona Onauskienė", "amzius": 70, "pajamos": 1500, "kredito\_balas": 620},

 {"vardas": "Aistė Aistytė", "amzius": 25, "pajamos": 900, "kredito\_balas": 550},

]

# Vertiname kiekvieno gyventojo paskolos galimybę

for gyventojas in gyventojai:

 print(vertinti\_paskola(gyventojas))

Pradinė Python programa (joje naudojamos **klasės**):

class Gyventojas:
 def \_\_init\_\_(self, vardas, amzius, pajamos, kredito\_balas):
 # čia turite surašyti klasės laukus

def vertinti\_paskola(gyventojas):
 # čia turite surašyti paskolos vertinimo kodą pagal duotus kriterijus

# Sukuriame gyventojų sąrašą
gyventojai = [
 Gyventojas("Jonas Jonauskas", 20, 2200, 700)
 # Čia turite papildyti testavimo duomenis
]

# Vertiname kiekvieno gyventojo paskolos galimybę
for g in gyventojai:
 print(vertinti\_paskola(g))

#### Vertinimas

| Nr. | Kas vertinama | Balai |
| --- | --- | --- |
| 1. | Yra testavimo duomenų rinkinys, tai paskola suteikiama | 1 |
| 2. | Teisingai surašyti gyventojo klasės laukai | 1 |
| 3. | Teisingai surašyta paskolos vertinimo logika | 2 |
| 4. | Yra testavimo duomenų rinkiniai kiekvieno kriterijaus vidinėms reikšmėms (ekvivalentinės klasės) | 2 |
| 5. | Yra testavimo duomenų rinkiniai kiekvieno kriterijaus ribinėms reikšmėms (ribinės reikšmės) | 2 |
| 6. | Pateikta programa su testinių duomenų rinkiniu veikia teisingai | 2 |
|  | Viso: | 10 |

#### Teisingas atsakymas

C++ sprendimas:

# include <iostream>
# include <string>
using namespace std;

struct Gyventojas {
 int amzius;
 double pajamos;
 int kredito\_balas;
 string vardas;
};

string vertinti\_paskola(Gyventojas gyventojas) {
 if (gyventojas.amzius < 18) {
 return gyventojas.vardas + " neatitinka amžiaus kriterijaus.";
 } else if (gyventojas.amzius > 65) {
 return gyventojas.vardas + " per didelis amžius.";
 } else if (gyventojas.pajamos < 1000) {
 return gyventojas.vardas + " pajamos per mažos.";
 } else if (gyventojas.kredito\_balas < 600) {
 return gyventojas.vardas + " kredito balas per žemas.";
 } else {
 return gyventojas.vardas + " paskola gali būti suteikta.";
 }
}

int main() {
 Gyventojas gyventojai[] = {
 // ekvivalentinės klasės
 {20, 2200, 700, "Jonas"} // Tinkamas variantas
 {30, 950, 650, "Ona"}, // Netinkamos pajamos
 {25, 1200, 590, "Petras"}, // Žemas kredito balas
 {15, 1500, 620, "Eglė"}, // Per mažas amžius
 {70, 2000, 700, "Antanas"} // Per didelis amžius
 // ribinės reikšmės
 {17, 2200, 700, "Marija"} // Neatitinka amžiaus
 {18, 2200, 700, "Nijolė"} // Atitinka amžių
 {66, 2200, 700, "Saulius"} // Neatitinka amžiaus
 {65, 2200, 700, "Darius"} // Atitinka amžių
 {30, 999, 650, "Gerda"}, // Netinkamos pajamos
 {30, 1000, 650, "Dana"}, // Tinkamos pajamos
 {25, 1200, 599, "Simas"}, // Netinkamas kreditas
 {25, 1200, 600, "Sonata"}, // Tinkamas kreditas
 };

 for (int i = 0; i < gyventojai.size(); ++i) {
 cout << vertinti\_paskola(gyventojai[i]) << endl;
 }

 return 0;
}

Python sprendimas

def vertinti\_paskola(gyventojas):

 if gyventojas["amzius"] < 18:

 return gyventojas["vardas"] + " neatitinka amžiaus kriterijaus."

 elif gyventojas["amzius"] > 65:

 return gyventojas["vardas"] + " per didelis amžius."

 elif gyventojas["pajamos"] < 1000:

 return gyventojas["vardas"] + " pajamos per mažos."

 elif gyventojas["kredito\_balas"] < 600:

 return gyventojas["vardas"] + " kredito balas per žemas."

 else:

 return gyventojas["vardas"] + " paskola gali būti suteikta."

# Sukuriame gyventojų sąrašą (kaip žodynus)

gyventojai = [

 {"vardas": "Jonas", "amzius": 20, "pajamos": 2200, "kredito\_balas": 700}, # Tinkamas variantas

 {"vardas": "Ona", "amzius": 30, "pajamos": 950, "kredito\_balas": 650}, # Netinkamos pajamos

 {"vardas": "Petras", "amzius": 25, "pajamos": 1200, "kredito\_balas": 590}, # Žemas kredito balas

 {"vardas": "Eglė", "amzius": 15, "pajamos": 1500, "kredito\_balas": 620}, # Per mažas amžius

 {"vardas": "Antanas", "amzius": 70, "pajamos": 2000, "kredito\_balas": 700}, # Per didelis amžius

 # Ribinės reikšmės

 {"vardas": "Marija", "amzius": 17, "pajamos": 2200, "kredito\_balas": 700}, # Neatitinka amžiaus

 {"vardas": "Nijolė", "amzius": 18, "pajamos": 2200, "kredito\_balas": 700}, # Atitinka amžių

 {"vardas": "Saulius", "amzius": 66, "pajamos": 2200, "kredito\_balas": 700}, # Neatitinka amžiaus

 {"vardas": "Darius", "amzius": 65, "pajamos": 2200, "kredito\_balas": 700}, # Atitinka amžių

 {"vardas": "Gerda", "amzius": 30, "pajamos": 999, "kredito\_balas": 650}, # Netinkamos pajamos

 {"vardas": "Dana", "amzius": 30, "pajamos": 1000, "kredito\_balas": 650}, # Tinkamos pajamos

 {"vardas": "Simas", "amzius": 25, "pajamos": 1200, "kredito\_balas": 599}, # Netinkamas kredito balas

 {"vardas": "Sonata", "amzius": 25, "pajamos": 1200, "kredito\_balas": 600} # Tinkamas kredito balas

]

# Vertiname kiekvieno gyventojo paskolos galimybę

for g in gyventojai:

 print(vertinti\_paskola(g))

Python sprendimas (2 var.; jame naudojamos **klasės**):

class Gyventojas:
 def \_\_init\_\_(self, vardas, amzius, pajamos, kredito\_balas):
 self.vardas = vardas
 self.amzius = amzius
 self.pajamos = pajamos
 self.kredito\_balas = kredito\_balas

def vertinti\_paskola(gyventojas):
 if gyventojas.amzius < 18:
 return gyventojas.vardas + " neatitinka amžiaus kriterijaus."
 elif gyventojas.amzius > 65:
 return gyventojas.vardas + " per didelis amžius."
 elif gyventojas.pajamos < 1000:
 return gyventojas.vardas + " pajamos per mažos."
 elif gyventojas.kredito\_balas < 600:
 return gyventojas.vardas + " kredito balas per žemas."
 else:
 return gyventojas.vardas + " paskola gali būti suteikta."

# Sukuriame gyventojų sąrašą
gyventojai = [
 Gyventojas("Jonas", 20, 2200, 700), # Tinkamas variantas
 Gyventojas("Ona", 30, 950, 650), # Netinkamos pajamos
 Gyventojas("Petras", 25, 1200, 590), # Žemas kredito balas
 Gyventojas("Eglė", 15, 1500, 620), # Per mažas amžius
 Gyventojas("Antanas", 70, 2000, 700), # Per didelis amžius
 # Ribinės reikšmės
 Gyventojas("Marija", 17, 2200, 700), # Neatitinka amžiaus
 Gyventojas("Nijolė", 18, 2200, 700), # Atitinka amžių
 Gyventojas("Saulius", 66, 2200, 700), # Neatitinka amžiaus
 Gyventojas("Darius", 65, 2200, 700), # Atitinka amžių
 Gyventojas("Gerda", 30, 999, 650), # Netinkamos pajamos
 Gyventojas("Dana", 30, 1000, 650), # Tinkamos pajamos
 Gyventojas("Simas", 25, 1200, 599), # Netinkamas kredito balas
 Gyventojas("Sonata", 25, 1200, 600) # Tinkamas kredito balas
]

# Vertiname kiekvieno gyventojo paskolos galimybę
for g in gyventojai:
 print(vertinti\_paskola(g))

#### Taškų skaičius

10

#### Sudėtingumo lygis

4

#### Ugdomos kompetencijos

Pažinimo