Code Academy

1 LYGIS

16 paskaita.
UNIT testų kūrimas

2023

Aurimas Aleksandras Nausėdas

Šiandien išmoksite

01 Susipažinsite su testų rašymo privalumais

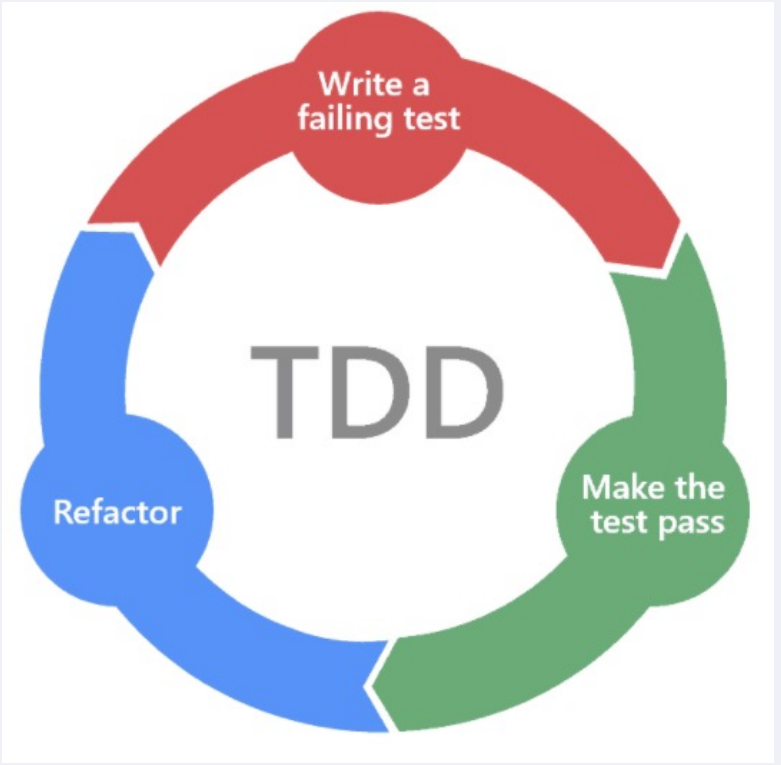
02 Naudoti unittest biblioteką

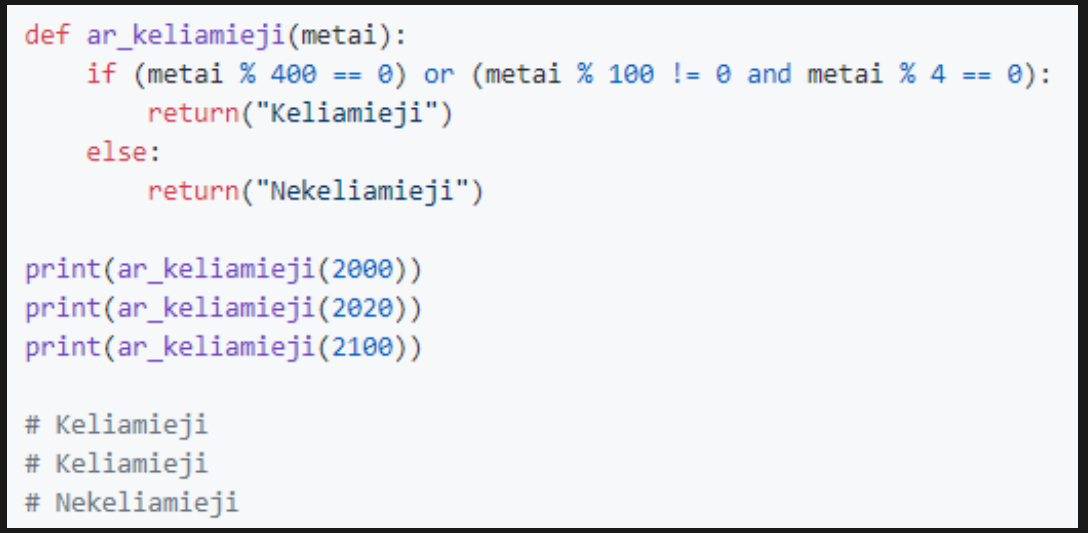
UNIT testų privalumai

- Galimybė išvengti klaidų rašant ar taisant kodą
- UNIT testai gali būti panaudoti kaip būsimos programos dokumentacija
- Sutaupo laiko testuotojų komandai
- Taupo pinigus (klaidų taisymas vėliau yra brangus)

Testavimu paremtas programavimas (TDD)

Iš pradžių sukuriame testą – po to parašome kodą





```python
def ar_keliamieji(metai):
    if (metai % 400 == 0) or (metai % 100 != 0 and metai % 4 == 0):
        return("Keliamieji")
    else:
        return("Nekeliamieji")

print(ar_keliamieji(2000))
print(ar_keliamieji(2020))
print(ar_keliamieji(2100))

# Keliamieji
# Keliamieji
# Nekeliamieji
```

Kaip patikrinti, ar programa teisingai veikia

Pavyzdys kaip darome dabar faile keliamieji.py

```python
import unittest
from keliamieji import *

class TestKeliamieji(unittest.TestCase):

    def test_ar_keliamieji(self):
        rezultatas = ar_keliamieji(2000)
        lukestis = "Keliamieji"
        self.assertEqual(lukestis, rezultatas)

# Ran 1 test in 0.007s
# OK
```

Kaip ištestuoti programą UNIT testų pagalba

Pavyzdys naujame faile test_keliamieji.py

```
python -m unittest test_keliamieji.py

.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Testo paleidimas komandinėje eilutėje

```
if __name__ == '__main__':
    unittest.main()
```

```
python test_keliamieji.py
```

Testo paleidimas tiesiogiai

Faile test_keliamieji.py prirašyti '__main__', tada UNIT test failą, galime leisti tiesiogiai

```python
import unittest
from keliamieji import ar_keliamieji
class TestKeliamieji(unittest.TestCase):
    def test_ar_keliamieji(self):
        self.assertEqual("Keliamieji", ar_keliamieji(2000))
        self.assertEqual("Keliamieji", ar_keliamieji(2020))
        self.assertEqual("Keliamieji", ar_keliamieji(2100))


# Ran 1 test in 0.003s

# FAILED (failures=1)


# Nekeliamieji != Keliamieji

# Expected :Keliamieji
# Actual   :Nekeliamieji
```

Pats testas turi būti teisingas

Metodas	Tikrina	Python versija nuo
assertEqual(a, b)	a == b	
assertNotEqual(a, b)	a != b	
assertTrue(x)	bool(x) is True	
assertFalse(x)	bool(x) is False	
assertIs(a, b)	a is b	3.1
assertIsNot(a, b)	a is not b	3.1
assertIsNone(x)	x is None	3.1
assertIsNotNone(x)	x is not None	3.1
assertIn(a, b)	a in b	3.1
assertNotIn(a, b)	a not in b	3.1
assertIsInstance(a, b)	isinstance(a, b)	3.2
assertNotIsInstance(a, b)	not isinstance(a, b)	3.2

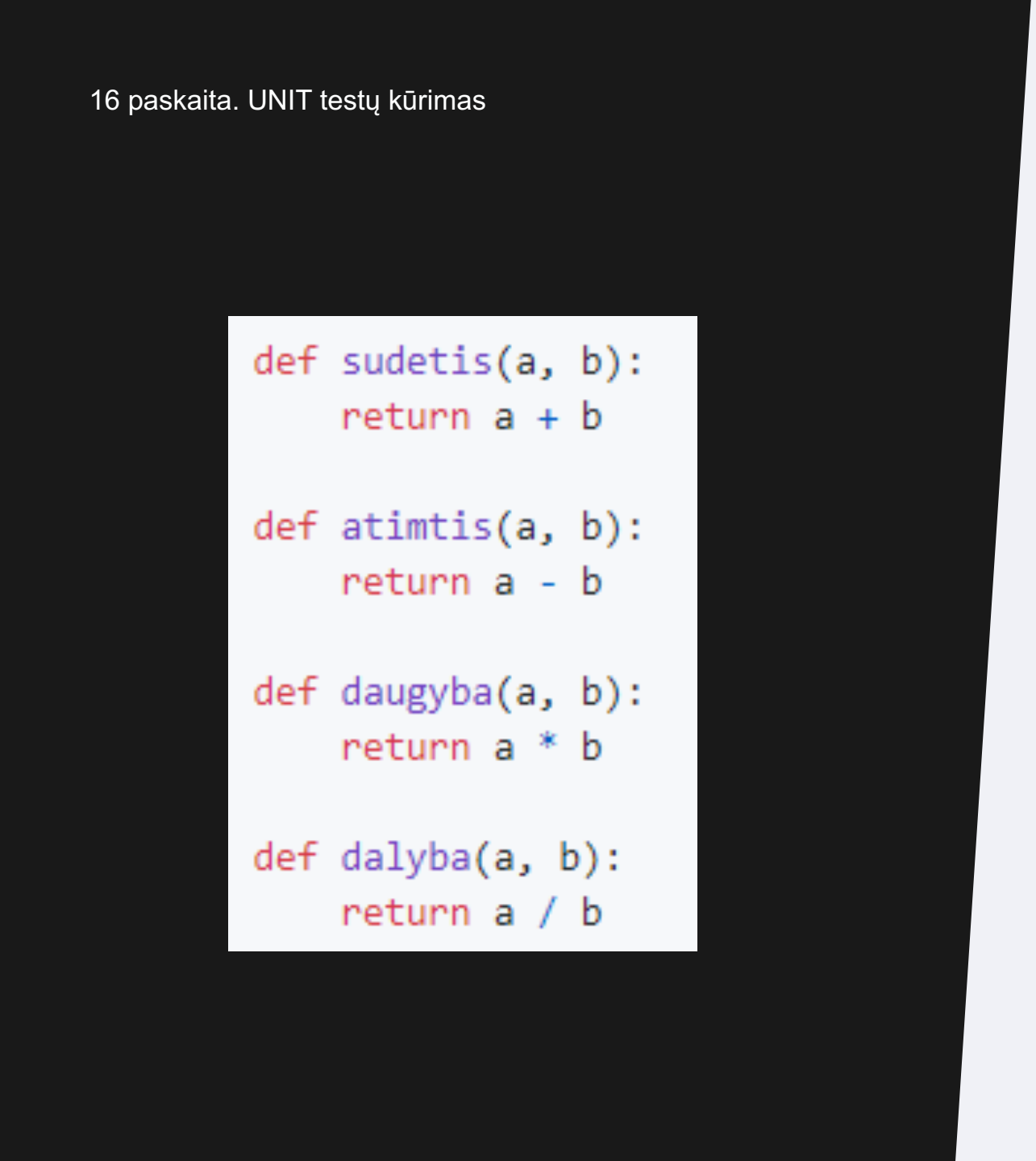
UNIT testų metodai

```python
def ar_keliamieji(metai):
    if (metai % 400 == 0) or (metai % 4 == 0):
        return("Keliamieji")
    else:
        return("Nekeliamieji")
```

```
Ran 1 test in 0.003s

FAILED (failures=1)

Keliamieji != Nekeliamieji

Expected :Nekeliamieji
Actual   :Keliamieji
```

Kai yra klaida kode

Faile keliamieji.py



```python
def sudetis(a, b):
    return a + b

def atimtis(a, b):
    return a - b

def daugyba(a, b):
    return a * b

def dalyba(a, b):
    return a / b
```

Kai yra klaida kode

Failas aritmetika.py

```python
import unittest
import aritmetika


class TestAritmetika(unittest.TestCase):
    def test_sudetis(self):
        self.assertEqual(15, aritmetika.sudetis(10, 5))
        self.assertEqual(0, aritmetika.sudetis(-1, 1))
        self.assertEqual(-2, aritmetika.sudetis(-1, -1))

    def test_atimtis(self):
        self.assertEqual(5, aritmetika.atimtis(10, 5))
        self.assertEqual(-2, aritmetika.atimtis(-1, 1))
        self.assertEqual(0, aritmetika.atimtis(-1, -1))

    def test_daugyba(self):
        self.assertEqual(50, aritmetika.daugyba(10, 5))
        self.assertEqual(-1, aritmetika.daugyba(-1, 1))
        self.assertEqual(1, aritmetika.daugyba(-1, -1))

    def test_dalyba(self):
        self.assertEqual(2, aritmetika.dalyba(10, 5))
        self.assertEqual(-1, aritmetika.dalyba(-1, 1))
        self.assertEqual(1, aritmetika.dalyba(-1, -1))


# Ran 4 tests in 0.002s

# OK
```

Kai yra klaida kode

Failas test_aritmetika.py

```python
def sudetis(a, b):
    return a + b

def atimtis(a, b):
    return a - b

def daugyba(a, b):
    return a ** b

def dalyba(a, b):
    return a / b
```

```
Ran 4 tests in 0.004s

FAILED (failures=1)


100000 != 50

Expected :50
Actual   :100000
```

Kai yra klaida kode

Failas aritmetika.py, šiuo atveju testas parodo klaidą, todėl ją galime nesudėtingai ištaisyti

```python
def sudetis(a, b):
    return a + b

def atimtis(a, b):
    return a - b

def daugyba(a, b):
    return a * b

def dalyba(a, b):
    return a // b
```

```
Ran 4 tests in 0.002s

OK
```

Kai yra klaida kode

Failas aritmetika.py, šiuo atveju testas neparodo klaidos, todėl turime koreguoti testą

```python
def test_dalyba(self):
    self.assertEqual(2, aritmetika.dalyba(10, 5))
    self.assertEqual(-1, aritmetika.dalyba(-1, 1))
    self.assertEqual(1, aritmetika.dalyba(-1, -1))
    self.assertEqual(2.5, aritmetika.dalyba(5, 2))
```

```
2 != 2.5

Expected :2.5
Actual   :2
```

Kai yra klaida kode

Failas test_aritmetika.py

```python
def test_dalyba(self):
    self.assertEqual(2, aritmetika.dalyba(10, 5))
    self.assertEqual(-1, aritmetika.dalyba(-1, 1))
    self.assertEqual(1, aritmetika.dalyba(-1, -1))
    self.assertEqual(2.5, aritmetika.dalyba(5, 2))
    self.assertRaises(ZeroDivisionError, aritmetika.dalyba, 10, 0)
```

```python
def test_dalyba(self):
    self.assertEqual(2, aritmetika.dalyba(10, 5))
    self.assertEqual(-1, aritmetika.dalyba(-1, 1))
    self.assertEqual(1, aritmetika.dalyba(-1, -1))
    self.assertEqual(2.5, aritmetika.dalyba(5, 2))
    with self.assertRaises(ZeroDivisionError):
        aritmetika.dalyba(10, 0)
```

```
Ran 4 tests in 0.002s

OK
```

Kaip patikrinti, ar kodas išmeta reikiamą klaidą

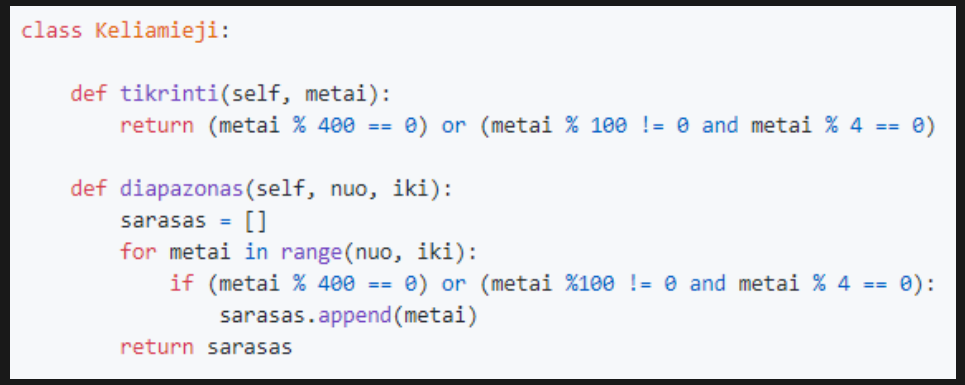
Failas test_aritmetika.py

```python
def ar_keliamieji2(metai):
    return (metai % 400 == 0) or (metai % 100 !=0 and metai % 4 == 0)
```

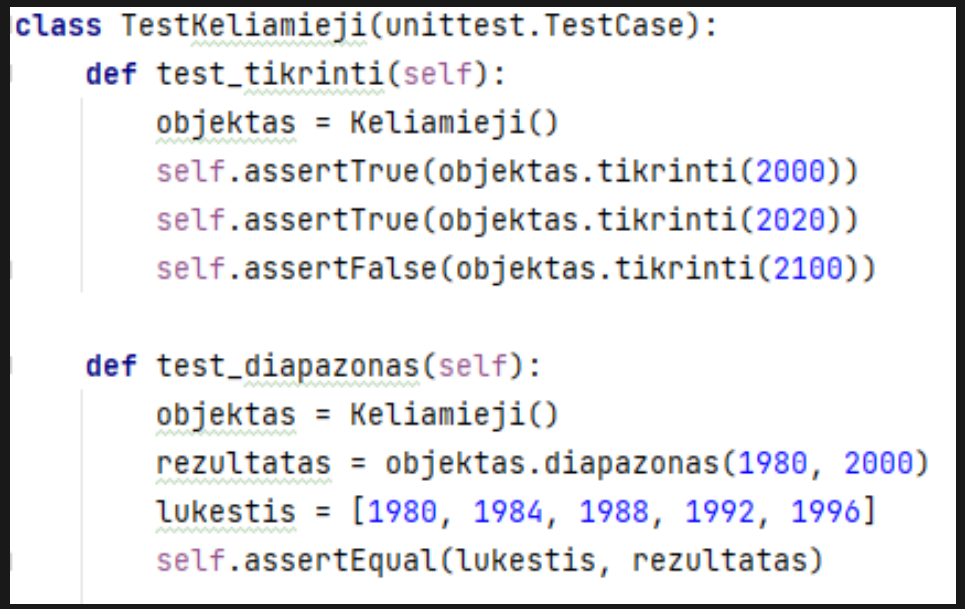
```python
import unittest
from keliamieji import ar_keliamieji2

class TestKeliamieji2(unittest.TestCase):
    def test_ar_keliamieji(self):
        self.assertTrue(ar_keliamieji2(2000))
        self.assertTrue(ar_keliamieji2(2020))
        self.assertFalse(ar_keliamieji2(2100))
```

Kaip ištestuoti gražinamas True/False reikšmes

- Faile keliamieji.py įrašome naują funkciją
- Faile test_keliamieji.py pridedame naują testą



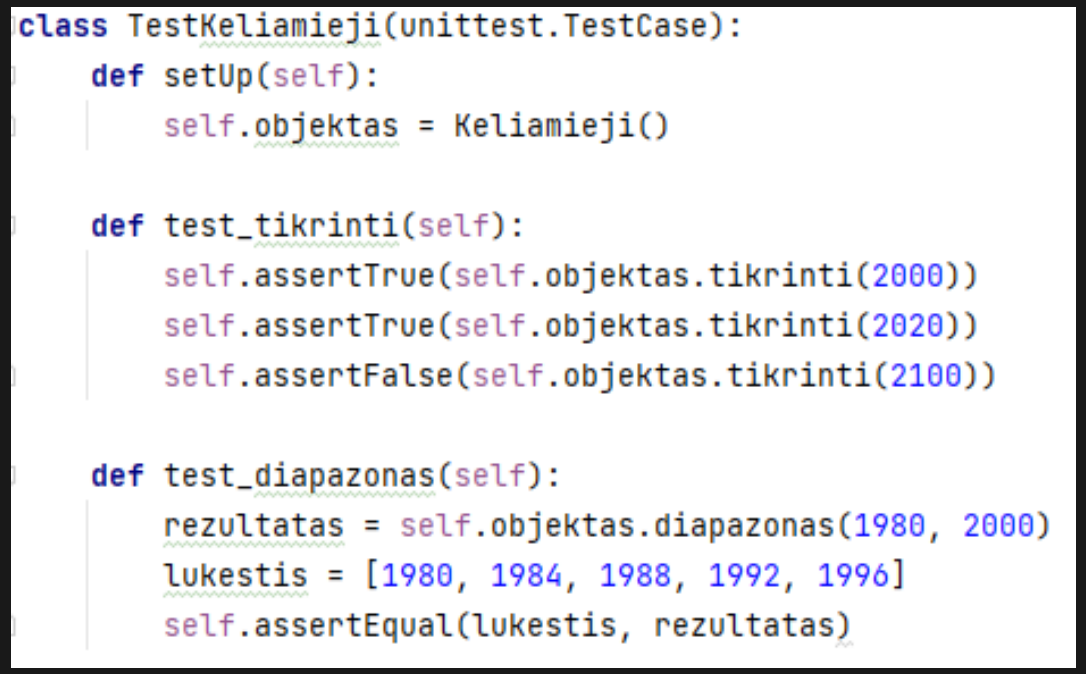
```python
class Keliamieji:

    def tikrinti(self, metai):
        return (metai % 400 == 0) or (metai % 100 != 0 and metai % 4 == 0)

    def diapazonas(self, nuo, iki):
        sarasas = []
        for metai in range(nuo, iki):
            if (metai % 400 == 0) or (metai %100 != 0 and metai % 4 == 0):
                sarasas.append(metai)
        return sarasas
```



```python
class TestKeliamieji(unittest.TestCase):
    def test_tikrinti(self):
        objektas = Keliamieji()
        self.assertTrue(objektas.tikrinti(2000))
        self.assertTrue(objektas.tikrinti(2020))
        self.assertFalse(objektas.tikrinti(2100))

    def test_diapazonas(self):
        objektas = Keliamieji()
        rezultatas = objektas.diapazonas(1980, 2000)
        lukestis = [1980, 1984, 1988, 1992, 1996]
        self.assertEqual(lukestis, rezultatas)
```

Objektų klasių testavimas

- Faile keliamieji.py įrašome naują klasę
- Faile test_keliamieji.py pridedame naują testą



```python
class TestKeliamieji(unittest.TestCase):
    def setUp(self):
        self.objektas = Keliamieji()

    def test_tikrinti(self):
        self.assertTrue(self.objektas.tikrinti(2000))
        self.assertTrue(self.objektas.tikrinti(2020))
        self.assertFalse(self.objektas.tikrinti(2100))

    def test_diapazonas(self):
        rezultatas = self.objektas.diapazonas(1980, 2000)
        lukestis = [1980, 1984, 1988, 1992, 1996]
        self.assertEqual(lukestis, rezultatas)
```

Patogesnis būdas

Faile test_keliamieji.py pridedame naują "setUp" metodą, kuriame sukuriame klasę kurią testuosime

Užduotis nr. 1

Pasiimti anksčiau sukurtą programos kodą (iš Teams)

- Funkcijas perdaryti taip, kad jos gražintų duomenis
- Sukurti UNIT testą visoms funkcijoms
- Kiekvienai funkcijai turi būti mažiausiai 3 patikrinimai
- Maksimaliai patobulinti kodą, nuolatos leidžiant sukurtą UNIT testą

Užduotis nr. 2

Pasiimti anksčiau sukurtą programos kodą (iš Teams)

- Teste sukurti setUp() metodą, kuriame būtų inicijuotas klasės objektas
- Funkcijas perdaryti taip, kad jos gražintų duomenis
- Sukurti UNIT testą visoms funkcijoms
- Kiekvienai funkcijai turi būti mažiausiai 3 patikrinimai
- Maksimaliai patobulinti kodą, nuolatos leidžiant sukurtą UNIT testą

Užduotis nr. 3

Nuosekliai, papunkčiui, pagal duotą UNIT testą sukurti programą, skaičiuojančią KMI:

```
import unittest
from kmi_skaiciavimas import kmi
class TestSkaiciavimas(unittest.TestCase):
    def test_kmi(self):
        self.assertEqual(kmi(78, 1.82), 23.54788069073783)
        self.assertEqual(kmi(50, 1.56), 20.5456936226167)
        self.assertEqual(kmi(100, 1.90), 27.70083102493075)
        with self.assertRaises(ValueError):
            kmi(20, 1.40)
        with self.assertRaises(ValueError):
            kmi(240, 1.40)
        with self.assertRaises(ValueError):
            kmi(80, 0.40)
        with self.assertRaises(ValueError):
            kmi(80, 3.40)
```

Namų darbas

Užbaigti klasėje nepadarytas užduotis

Išspręsti paskaitos uždaviniai (įkelti ketvirtadienį)

https://github.com/aurimas13/Python-Beginner-Course/tree/main/Programs

Naudinga informacija