Евменов Владимир

ИУ5-32Б

***Рубежный контроль №2***

*Задание:*

*1) Проведите рефакторинг текста программы рубежного контроля №1 таким образом, чтобы он был пригоден для модульного тестирования.*

*2) Для текста программы рубежного контроля №1 создайте модульные тесты с применением TDD - фреймворка (3 теста).*

Код программы рк1:

class House:

def init(self, house\_id, street\_id, name):

self.house\_id = house\_id

self.street\_id = street\_id

self.name = name

class Street:

def init(self, street\_id, name):

self.street\_id = street\_id

self.name = name

# Тестовые данные

houses = [

House(1, 1, "46"),

House(2, 1, "82"),

House(3, 2, "3"),

House(4, 3, "51"),

House(5, 3, "16")

]

streets = [

Street(1, "Молодёжная"),

Street(2, "Бауманская"),

Street(3, "Ландышевая")

]

# Задание 1

result1 = [(house.name, street.name) for house in houses for street in streets if house.street\_id == street.street\_id]

result1.sort(key=lambda x: x[0])

print("Список всех связанных домов и улиц, отсортированный по домам:")

print(result1)

# Задание 2

result2 = [(street.name, sum(1 for house in houses if house.street\_id == street.street\_id)) for street in streets]

result2.sort(key=lambda x: x[1])

print("Список улиц с домами, отсортированный по количеству домов:")

print(result2)

# Задание 3

result3 = [(house.name, street.name) for house in houses for street in streets if house.street\_id == street.street\_id and house.name.endswith("я")]

print("Список всех домов, у которых название заканчивается на «я», и улицы, на которых они находятся:")

print(result3)

*Добавим конструкторы* ***\_\_init\_\_****, чтобы было возможно создавать отдельные экземпляры классов* ***House*** *и* ***Street****:*

class House:

def \_\_init\_\_(self, house\_id, street\_id, name):

self.house\_id = house\_id

self.street\_id = street\_id

self.name = name

class Street:

def \_\_init\_\_(self, street\_id, name):

self.street\_id = street\_id

self.name = name

def find\_houses\_and\_streets(houses, streets):

return [(house.name, street.name) for house in houses for street in streets if house.street\_id == street.street\_id]

def count\_houses\_in\_streets(houses, streets):

return [(street.name, sum(1 for house in houses if house.street\_id == street.street\_id)) for street in streets]

def find\_specific\_houses(houses, streets):

return [(house.name, street.name) for house in houses for street in streets if house.street\_id == street.street\_id and house.name.endswith("я")]

*Теперь для данной обработки данных в формате функций напишем модульные тесты с использованием TDD-фреймворков, таких как* ***unittest*** *или* ***pytest****.*

import unittest

class TestHousesAndStreets(unittest.TestCase):

def setUp(self):

self.houses = [House(1, 1, "46"), House(2, 1, "82")]

self.streets = [Street(1, "Молодёжная"), Street(2, "Бауманская")]

def test\_find\_houses\_and\_streets(self):

result = find\_houses\_and\_streets(self.houses, self.streets)

self.assertEqual(result, [("46", "Ландышевая"), ("82", "Бауманская")])

def test\_count\_houses\_in\_streets(self):

result = count\_houses\_in\_streets(self.houses, self.streets)

self.assertEqual(result, [("Бауманская", 2), ("Молодежная", 0)])

def test\_find\_specific\_houses(self):

result = find\_specific\_houses(self.houses, self.streets)

self.assertEqual(result, [("46", "Молодежная")])

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

unittest.main()