Рубежный контроль представляет собой разработку тестов на языке Python
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).

Текст программы

```
import unittest
from operator import itemgetter

# ----------------------------------------
# МОДЕЛИ
# ----------------------------------------

class Schoolboy:
    """Информация об ученике."""
    def __init__(self, id, surname, mark, class_id):
        self.id = id
        self.surname = surname
        self.mark = mark
        self.class_id = class_id

class Class:
    """Информация о классе."""
    def __init__(self, id, name):
        self.id = id
        self.name = name

class SchoolboyClass:
    """
    Промежуточная сущность для связи «многие ко многим»
    между учениками и классами.
    """
    def __init__(self, class_id, schoolboy_id):
        self.class_id = class_id
        self.schoolboy_id = schoolboy_id

# ----------------------------------------
# ДАННЫЕ
# ----------------------------------------

classes = [
    Class(1, '9А'),
    Class(2, '10Б'),
    Class(3, '11В'),
    Class(4, '9Г'),
    Class(5, '10Д'),
]

schoolboys = [
    Schoolboy(1, 'Иванов',   5, 1),
    Schoolboy(2, 'Петров',   4, 2),
```

```
    Schoolboy(3, 'Сидоров',  3, 3),
    Schoolboy(4, 'Кузнецов', 5, 4),
    Schoolboy(5, 'Смирнов',  4, 5),
]

schoolboys_classes = [
    SchoolboyClass(1, 1),
    SchoolboyClass(2, 2),
    SchoolboyClass(3, 3),
    SchoolboyClass(4, 4),
    SchoolboyClass(5, 5),
    SchoolboyClass(1, 2),
    SchoolboyClass(2, 3),
    SchoolboyClass(3, 4),
    SchoolboyClass(4, 5),
    SchoolboyClass(5, 1),
]

# --------------------------------------------
# ФУНКЦИИ ДЛЯ РАСЧЁТОВ
# --------------------------------------------

def get_one_to_many(classes, schoolboys):
    """Формирует список (фамилия, оценка, название_класса)
       для связи один-ко-многим (Schoolboy -> Class)."""
    return [
        (s.surname, s.mark, c.name)
        for c in classes
        for s in schoolboys
        if s.class_id == c.id
    ]

def get_many_to_many(classes, schoolboys_classes, schoolboys):
    """Формирует список (фамилия, оценка, название_класса)
       для связи многие-ко-многим (через SchoolboyClass)."""
    many_to_many_temp = [
        (c.name, sc.class_id, sc.schoolboy_id)
        for c in classes
        for sc in schoolboys_classes
        if c.id == sc.class_id
    ]
    return [
        (s.surname, s.mark, class_name)
        for class_name, class_id, schoolboy_id in many_to_many_temp
        for s in schoolboys
        if s.id == schoolboy_id
    ]

def task_b1(classes, schoolboys):
    """
    B1: Сформировать список пар (фамилия, название класса)
        для всех, чьи фамилии оканчиваются на 'ов'.
```

```
    """
    return [
        (s.surname, c.name)
        for c in classes
        for s in schoolboys
        if s.class_id == c.id and s.surname.endswith('ов')
    ]

def task_b2(classes, one_to_many):
    """
    B2: Сформировать список пар (название класса, средняя оценка)
        по всем классам и отсортировать их по возрастанию средней оценки.
    """
    res_12_unsorted = []
    for c in classes:
        # Берём всех учеников, которые числятся в данном классе (через one_to_many)
        c_schoolboys = list(filter(lambda x: x[2] == c.name, one_to_many))
        if c_schoolboys:  # если в классе есть ученики
            marks = [mark for _, mark, _ in c_schoolboys]
            avg_mark = sum(marks) / len(marks)
            res_12_unsorted.append((c.name, avg_mark))

    # Сортируем по второй позиции (средняя оценка)
    return sorted(res_12_unsorted, key=itemgetter(1))

def task_b3(many_to_many):
    """
    B3: Отсортировать список (фамилия, оценка, название класса)
        по фамилиям (алфавитно).
    """
    return sorted(many_to_many, key=itemgetter(0))

# --------------------------------------------
# ОСНОВНАЯ ФУНКЦИЯ (при желании запускать)
# --------------------------------------------

def main():
    one_to_many = get_one_to_many(classes, schoolboys)
    many_to_many = get_many_to_many(classes, schoolboys_classes, schoolboys)

    print("Task B1")
    print(task_b1(classes, schoolboys))

    print("\nTask B2")
    print(task_b2(classes, one_to_many))

    print("\nTask B3")
    print(task_b3(many_to_many))

# --------------------------------------------
# ТЕСТЫ (TDD, unittest)
# --------------------------------------------
```

```
class TestSchoolboyData(unittest.TestCase):
    """Набор тестов для проверки задач B1, B2, B3."""

    def setUp(self):
        """Подготавливаем данные, которые нужны в каждом тесте."""
        self._classes = classes
        self._schoolboys = schoolboys
        self._schoolboys_classes = schoolboys_classes
        self._one_to_many = get_one_to_many(self._classes, self._schoolboys)
        self._many_to_many = get_many_to_many(self._classes,
                                              self._schoolboys_classes,
                                              self._schoolboys)

    def test_task_b1(self):
        """Проверяем, что фамилии с окончанием 'ов' формируют правильные пары."""
        result = task_b1(self._classes, self._schoolboys)
        expected = [
            ('Иванов',   '9А'),
            ('Петров',   '10Б'),
            ('Сидоров',  '11В'),
            ('Кузнецов', '9Г'),
            ('Смирнов',  '10Д')
        ]
        self.assertEqual(result, expected)

    def test_task_b2(self):
        """Проверяем, что средние оценки по классам вычисляются и сортируются верно."""
        result = task_b2(self._classes, self._one_to_many)
        # Расчёт вручную:
        #   9А (Иванов - 5),  avg=5
        #   10Б (Петров - 4), avg=4
        #   11В (Сидоров -3), avg=3
        #   9Г (Кузнецов-5),  avg=5
        #   10Д (Смирнов -4), avg=4
        # Отсортированная последовательность (по возрастанию среднего балла):
        #    [ ('11В', 3), ('10Б', 4), ('10Д', 4), ('9А', 5), ('9Г', 5) ]
        expected = [
            ('11В', 3.0),
            ('10Б', 4.0),
            ('10Д', 4.0),
            ('9А',  5.0),
            ('9Г',  5.0)
        ]
        self.assertEqual(result, expected)

    def test_task_b3(self):
        """Проверяем, что результирующий список многие-ко-многим
           корректно сортируется по фамилиям (алфавит)."""
        result = task_b3(self._many_to_many)
        # В списке 5 учеников, каждый фигурирует в двух классах (см. schoolboys_classes),
        # итого 10 записей (surname, mark, classname). Проверим длину и первую фамилию:
```

```
        self.assertEqual(len(result), 10,
                         "Ожидается 10 элементов в списке many_to_many.")
        self.assertEqual(result[0][0], 'Иванов',
                         "Первым по алфавиту должен идти Иванов.")

if __name__ == '__main__':
    # Запуск либо основной функции, либо тестов:
    # 1) Для «обычного» запуска всех заданий:  main()
    # 2) Для тестов:  python -m unittest данный_файл.py
    unittest.main()
```