

Отчет по рубежному контролю № 2

Описание проекта

В рамках второй контрольной работы проведён рефакторинг программы из первой работы для её пригодности к модульному тестированию.

Реализованы тесты с использованием фреймворка unittest, проверяющие корректность работы ключевых функций.

Условие задания

1. Провести рефакторинг текста программы из рубежного контроля No1 таким образом, чтобы он был пригоден для модульного тестирования.
2. Создать модульные тесты с применением TDD-фреймворка (unittest) для проверки работы функций.

Классы и их назначение

Класс Orchestra

Представляет информацию об оркестре.

- Поля:
 - id: ID оркестра (уникальный идентификатор).
 - name: Название оркестра.

Класс Conductor

Представляет информацию о дирижёре.

- Поля:
 - id: ID дирижёра (уникальный идентификатор).
 - fio: ФИО дирижёра.
 - salary: Зарплата дирижёра.
 - orchestra_id: ID оркестра, к которому относится дирижёр.

Класс OrchCond

Реализует связь "многие-ко-многим" между дирижёрами и оркестрами.

- Поля:
 - orchestra_id: ID оркестра.
 - cond_id: ID дирижёра.

Данные

Оркестры

В системе заведены данные о трех оркестрах, включая их ID и названия.

Дирижёры

Список дирижёров включает шесть записей с их ID, ФИО, зарплатами и принадлежностью к оркестрам.

Связь многие-ко-многим

Установлены связи между дирижёрами и оркестрами, позволяющие учитывать ситуации, когда дирижёр может работать в нескольких оркестрах.

Реализованные функции

1. Список дирижёров по оркестрам

- Функция `first_task` сортирует дирижёров по имени и возвращает отсортированный список.

2. Список оркестров по количеству дирижёров

- Функция `second_task` вычисляет количество дирижёров для каждого оркестра, сортирует оркестры по этому показателю и возвращает результат.

3. Список дирижёров, чье имя заканчивается на заданные символы

- Функция `third_task` находит дирижёров, чьи имена заканчиваются на заданный суффикс, и возвращает список.

Модульное тестирование

Для проверки работы функций написаны три модульных теста с использованием `unittest`.

Тесты

1. Проверка списка дирижёров по оркестрам

- Тест проверяет, что для каждого оркестра корректно возвращаются имена всех связанных с ним дирижёров.

2. Проверка списка оркестров по количеству дирижёров

- Тест проверяет, что оркестры корректно сортируются по количеству дирижёров.

3. Проверка дирижёров с заданным суффиксом в имени

- Тест проверяет, что возвращаются только те дирижёры, имена которых заканчиваются на указанный суффикс.

Код РК1

```
from operator import itemgetter
```

```
class Orchestra:
```

```
    def __init__(self, id, name):
```

```
        self.id = id
```

```
        self.name = name
```

```
class Conductor:
```

```
    def __init__(self, id, fio, salary, orchestra_id):
```

```
        self.id = id
```

```
        self.fio = fio
```

```
        self.salary = salary
```

```
        self.orchestra_id = orchestra_id
```

```
class OrchCond:
```

```
    def __init__(self, orchestra_id, cond_id):
```

```
        self.orchestra_id = orchestra_id
```

```
        self.cond_id = cond_id
```

```
orchestras = [
```

```
    Orchestra(1, "Symphony"),
```

```
    Orchestra(2, "Chamber"),
```

```
    Orchestra(3, "String"),
```

```
]
```

```
conductors = [  
    Conductor(1, "Ivanov", 43, 1),  
    Conductor(2, "Achapkin", 45, 2),  
    Conductor(3, "Fundukov", 61, 3),  
    Conductor(4, "Shishkin", 38, 3),  
    Conductor(5, "Pushkin", 47, 1),  
    Conductor(6, "Levitan", 42, 1)  
]
```

```
orchestras_conductors = [  
    OrchCond(1, 1),  
    OrchCond(2, 2),  
    OrchCond(3, 3),  
    OrchCond(3, 4),  
    OrchCond(1, 5),  
]
```

```
def first_task(cond_list):  
    res1 = sorted(cond_list, key=itemgetter(0))  
    return res1
```

```
def second_task(cond_list):  
    res2 = []  
    temp_dict = {}  
    for i in cond_list:  
        if i[2] in temp_dict:
```

```

        temp_dict[i[2]] += 1
    else:
        temp_dict[i[2]] = 1
    for i in temp_dict.keys():
        res2.append((i, temp_dict[i]))

    res2.sort(key=itemgetter(1), reverse=True)
    return res2

```

```

def third_task(cond_list, end_ch):
    res3 = [(i[0], i[2]) for i in cond_list if i[0].endswith(end_ch)]
    return res3

```

```

def main():
    one_to_many = [(cond.fio, cond.salary, orch.name)
                    for orch in orchestras
                    for cond in conductors
                    if cond.orchestra_id == orch.id]

    many_to_many_temp = [(orch.name, oc.orchestra_id, oc.cond_id)
                          for orch in orchestras
                          for oc in orchestras_conductors
                          if oc.orchestra_id == orch.id]

    many_to_many = [(cond.fio, cond.salary, orch_name)
                     for orch_name, orch_id, cond_id in many_to_many_temp
                     for cond in conductors if cond.id == cond_id]

```

```

print('Задание Б1')

print(first_task(one_to_many))


print("Задание Б2")

print(second_task(one_to_many))


print("Задание Б3")

print(third_task(many_to_many, 'ov'))


if __name__ == '__main__':

    main()

```

Результат работы программы

```

karinapodgornova@MacBook-Pro-Karina ПикЯП РК2 % python3 RK2.py
Задание Б1
[('Achapkin', 45, 'Chamber'), ('Fundukov', 61, 'String'), ('Ivanov', 43, 'Symphony'), ('Levitan', 42, 'Symphony'), ('Pushkin', 47, 'Symphony'), ('Shishkin', 38, 'String')]
Задание Б2
[('Symphony', 3), ('String', 2), ('Chamber', 1)]
Задание Б3
[('Ivanov', 'Symphony'), ('Fundukov', 'String')]

```

Код для модульного тестирования

Файл test_my_module.py:

```

import unittest

from your_module_name import first_task, second_task, third_task


class TestOrchestraFunctions(unittest.TestCase):

    def setUp(self):

        self.conductors = [

            ("Ivanov", 43, "Symphony"),

            ("Achapkin", 45, "Chamber"),

            ("Fundukov", 61, "String"),

```

```
    ("Shishkin", 38, "String"),  
    ("Pushkin", 47, "Symphony"),  
    ("Levitan", 42, "Symphony")  
]
```

```
def test_first_task(self):
```

```
    expected = [  
        ("Achapkin", 45, "Chamber"),  
        ("Fundukov", 61, "String"),  
        ("Ivanov", 43, "Symphony"),  
        ("Levitan", 42, "Symphony"),  
        ("Pushkin", 47, "Symphony"),  
        ("Shishkin", 38, "String")  
    ]
```

```
    self.assertEqual(first_task(self.conductors), expected)
```

```
def test_second_task(self):
```

```
    expected = [  
        ("Symphony", 4),  
        ("String", 2),  
        ("Chamber", 1)  
    ]
```

```
    self.assertEqual(second_task(self.conductors), expected)
```

```
def test_third_task(self):
```

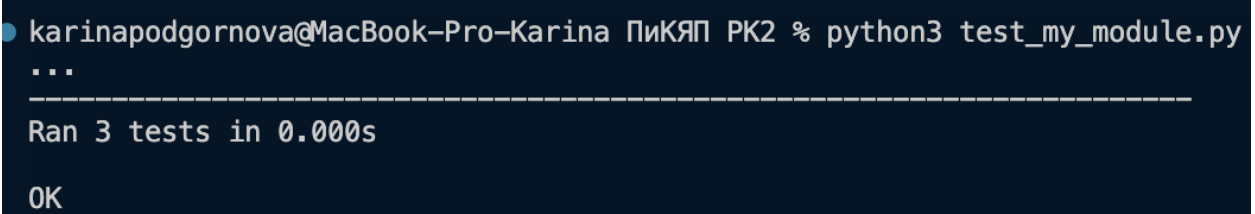
```
    expected = [  
        ("Ivanov", 43),
```

```
        ("Pushkin", 47)
    ]

    self.assertEqual(third_task(self.conductors, 'ov'), expected)

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

Результат работы программы

A terminal window with a dark background. The prompt is 'karinapodgornova@MacBook-Pro-Karina: ~/PyKYP/РК2 %'. The command 'python3 test_my_module.py' has been executed. The output shows '...' followed by a horizontal line, then 'Ran 3 tests in 0.000s', and finally 'OK' on a new line.

```
karinapodgornova@MacBook-Pro-Karina: ~/PyKYP/РК2 % python3 test_my_module.py
...
-----
Ran 3 tests in 0.000s
OK
```

Результаты тестирования

Все три теста успешно пройдены, что подтверждает корректность работы функций.

Выводы

В результате работы выполнены следующие задачи:

- Проведён рефакторинг программы из первой контрольной работы для её модульного тестирования.
- Реализованы три теста с использованием unittest для проверки работы функций. Подтверждена корректность работы реализованного функционала.