

**Московский государственный технический университет
им. Н.Э. Баумана**

Факультет информатика и системы управления
Кафедра системы обработки информации и управления

Курс «Парадигмы и конструкции языков программирования» Отчет по
рубежному контролю №1
Вариант Б17

Выполнил:
студент группы ИУ5-32Б:
Шпакова Д. Л.
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.
Подпись и дата:

Москва, 2025 г.

Постановка задачи

В рамках рубежного контроля №1 по курсу «Парадигмы и конструкции языков программирования» требовалось разработать программу на языке Python.

Цель работы:

1. Создать два класса данных, соответствующие предметной области «Дирижер» (Класс 1) и «Оркестр» (Класс 2) (Вариант №17).
2. Реализовать связь «один-ко-многим» между классами.
3. Создать дополнительный класс для реализации связи «многие-ко-многим».
4. Создать списки объектов классов, содержащие тестовые данные (3-5 записей).
5. Реализовать три запроса в соответствии с Вариантом Б, адаптировав их под выбранную предметную область.
6. При реализации запросов использовать функциональные возможности Python (list/dict comprehensions).

Задачи (Вариант Б):

1. «Оркестр» и «Дирижер» связаны соотношением «один-ко-многим». Вывести список всех связанных дирижеров и оркестров, отсортированный по дирижерам (по ФИО).
2. «Оркестр» и «Дирижер» связаны соотношением «один-ко-многим». Вывести список оркестров с количеством дирижеров в каждом, отсортированный по количеству дирижеров.
3. «Оркестр» и «Дирижер» связаны соотношением «многие-ко-многим». Вывести список всех дирижеров, у которых фамилия заканчивается на «ов», и названия их оркестров.

Текст программы

```
from operator import itemgetter

class Conductor:
    """Дирижер"""
    def __init__(self, id, fio, salary, orchestra_id):
        self.id = id
        self.fio = fio
        self.salary = salary # Количественный признак
        self.orchestra_id = orchestra_id # Для связи один-ко-многим

class Orchestra:
    """Оркестр"""
    def __init__(self, id, name):
        self.id = id
        self.name = name

class ConductorOrchestra:
    """
    'Дирижеры оркестра' для реализации связи многие-ко-многим
    """
    def __init__(self, conductor_id, orchestra_id):
        self.conductor_id = conductor_id
        self.orchestra_id = orchestra_id

# --- Тестовые данные ---

# Оркестры (Класс 2)
orchestras = [
    Orchestra(1, 'Венский филармонический'),
    Orchestra(2, 'Берлинский филармонический'),
    Orchestra(3, 'Лондонский симфонический'),

    # Дополнительные для многие-ко-многим
    Orchestra(11, 'Нью-Йоркский филармонический'),
    Orchestra(22, 'Бостонский симфонический'),
]

# Дирижеры (Класс 1)
conductors = [
    Conductor(1, 'Караян', 100000, 2),
    Conductor(2, 'Бернстайн', 120000, 1),
    Conductor(3, 'Мути', 90000, 1),
    Conductor(4, 'Петров', 80000, 3),
    Conductor(5, 'Иванов', 75000, 3),
]

# Данные для связи многие-ко-многим
conductors_orchestras = [
    ConductorOrchestra(1, 1), # Караян -> Венский
    ConductorOrchestra(1, 2), # Караян -> Берлинский
    ConductorOrchestra(2, 11), # Бернстайн -> Нью-Йоркский
    ConductorOrchestra(3, 1), # Мути -> Венский
```

```

ConductorOrchestra(4, 3), # Петров -> Лондонский
ConductorOrchestra(4, 22), # Петров -> Бостонский
ConductorOrchestra(5, 3), # Иванов -> Лондонский
ConductorOrchestra(5, 11), # Иванов -> Нью-Йоркский
]

def main():
    """Основная функция"""

    #Подготовка данных

    # Соединение данных один-ко-многим (Дирижер "принадлежит" одному Оркестру)
    # (Фамилия дирижера, Зарплата дирижера, Название оркестра)
    one_to_many = [(c.fio, c.salary, o.name)
                    for o in orchestras
                    for c in conductors
                    if c.orchestra_id == o.id]

    # Соединение данных многие-ко-многим
    many_to_many_temp = [(o.name, co.orchestra_id, co.conductor_id)
                          for o in orchestras
                          for co in conductors_orchestras
                          if o.id == co.orchestra_id]

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

    #Выполнение запросов

    print('Задание Б1')
    # "Выведите список всех связанных сотрудников и отделов,
    # отсортированный по сотрудникам..." (Дирижерам)
    res_b1 = sorted(one_to_many, key=itemgetter(0))
    print(res_b1)

    print('\nЗадание Б2')
    # "Выведите список отделов (Оркестров) с количеством
    # сотрудников (Дирижеров) в каждом отделе,
    # отсортированный по количеству сотрудников."

    res_b2_unsorted = []
    for o in orchestras:
        o_conds = list(filter(lambda i: i[2] == o.name, one_to_many))

        if len(o_conds) > 0:
            res_b2_unsorted.append((o.name, len(o_conds)))

    res_b2 = sorted(res_b2_unsorted, key=itemgetter(1), reverse=True)
    print(res_b2)

    print('\nЗадание Б3')
    # "Выведите список всех сотрудников (Дирижеров), у которых
    # фамилия заканчивается на «ов», и названия их отделов (Оркестров)."
```

```

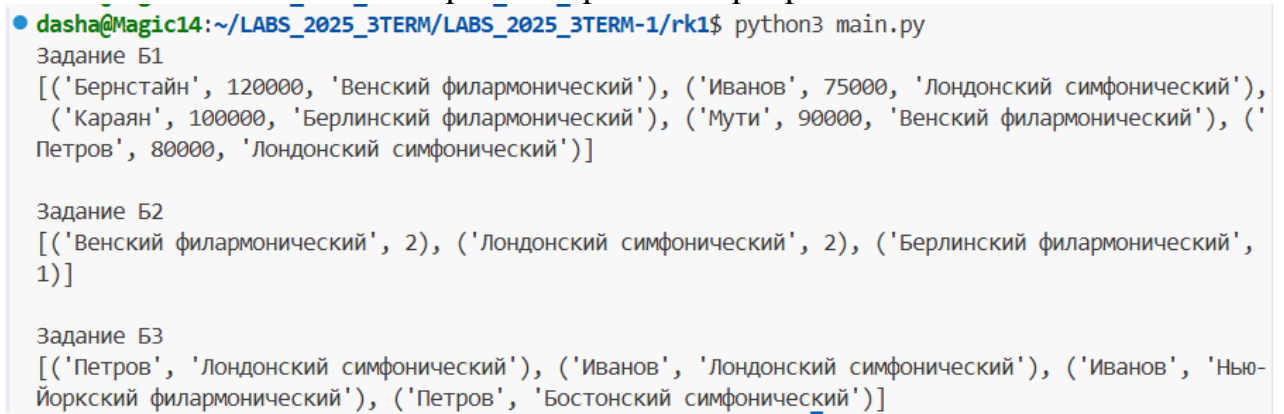
# (Используем связь многие-ко-многим)

res_b3 = [(fio, orch_name)
           for fio, _, orch_name in many_to_many
           if fio.endswith('ов')]
print(res_b3)

if __name__ == '__main__':
    main()

```

Скриншот работы программы



```

dasha@Magic14:~/LABS_2025_3TERM/LABS_2025_3TERM-1/rk1$ python3 main.py
Задание Б1
[('Бернштейн', 120000, 'Венский филармонический'), ('Иванов', 75000, 'Лондонский симфонический'),
 ('Караян', 100000, 'Берлинский филармонический'), ('Мути', 90000, 'Венский филармонический'), ('
Петров', 80000, 'Лондонский симфонический')]

Задание Б2
[('Венский филармонический', 2), ('Лондонский симфонический', 2), ('Берлинский филармонический',
1)]

Задание Б3
[('Петров', 'Лондонский симфонический'), ('Иванов', 'Лондонский симфонический'), ('Иванов', 'Нью-
Йоркский филармонический'), ('Петров', 'Бостонский симфонический')]

```

Рис. 1 Вывод результатов программы