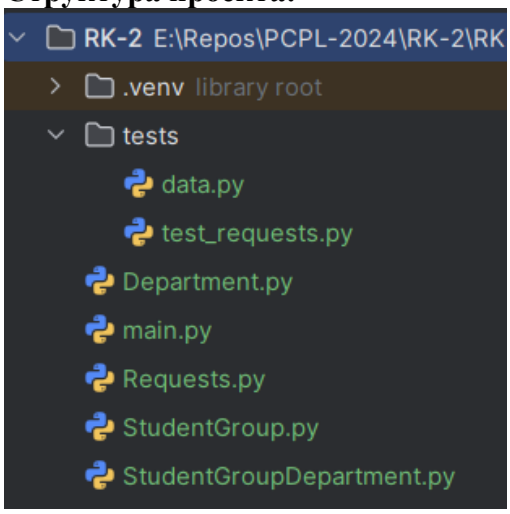
РК-2 Курс ПиКЯП Яншин Тимур ИУ5-36Б

Вариант 28Д

Структура проекта:



Текст программы:

main.py

```
from Requests import findGroupsEndWithKey,
DepartmentsSortedByAvgNumberOfStudents, findDepartmentsStartWithKey
from tests.data import get_departments, get_student_groups,
get_student_group_departments


def prepare_relations(departments, student_groups, group_deps):
    # Отношение один-ко-многим
    one_to_many = [
        (g.group_name, g.number_of_students, d.name)
        for d in departments
        for g in student_groups
        if g.department_id == d.id
    ]

    # Отношение многие-ко-многим
    temp = [
        (group.group_name, group_dep.department_id)
        for group in student_groups
        for group_dep in group_deps
        if group.id == group_dep.student_group_id
    ]

    many_to_many = [
        (group_name, dept.name)
        for dept in departments
        for group_name, dep_id in temp
        if dep_id == dept.id
    ]

    return one_to_many, many_to_many

def main():
    departments = get_departments()
    student_groups = get_student_groups()
    group_deps = get_student_group_departments()

    one_to_many, many_to_many = prepare_relations(departments,
```

```
student_groups, group_deps)

    # Выполнение запросов
    findGroupsEndWithKey(one_to_many, "03")
    DepartmentsSortedByAvgNumberOfStudents(one_to_many)
    findDepartmentsStartWithKey(many_to_many, "A")

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

Department.py

```
class Department:
    def __init__(self, id, name):
        self.id = id
        self.name = name

    def __repr__(self):
        return f"Department(id={self.id}, name='{self.name}')"
```

Requests.py

```
from collections import defaultdict

def findGroupsEndWithKey(one_to_many, key, debug=True):
    """
    Запрос 1: Студенческие группы, заканчивающиеся на key и их кафедры
    """
    filtered_groups = [
        {"group_name": group_name, "department_name": department_name}
        for group_name, _, department_name in one_to_many
        if group_name.endswith(key)
    ]

    if debug:
        print(f"Запрос 1: Студенческие группы, заканчивающиеся на '{key}' и
их кафедры:")
        if not filtered_groups:
            print("Ничего не найдено")
        else:
            for item in filtered_groups:
                print(f"Группа: {item['group_name']}, Кафедра:
{item['department_name']}")
        print("\n")

    return filtered_groups

def DepartmentsSortedByAvgNumberOfStudents(one_to_many, debug=True):
    """
    Запрос 2: Кафедры, отсортированные по среднему кол-ву студентов в группах
    """
    department_student_counts = defaultdict(list)
    for _, number_of_students, department_name in one_to_many:
        department_student_counts[department_name].append(number_of_students)

    average_students = [
        {
            "department_name": dept,
            "average_number_of_students": sum(counts) / len(counts)
        }
        for dept, counts in department_student_counts.items()
    ]
```

```
    sorted_average_students = sorted(
        average_students,
        key=lambda x: x["average_number_of_students"],
        reverse=True
    )

    if debug:
        print("Запрос 2: Кафедры со средним количеством студентов,
отсортированные по среднему количеству:")
        for item in sorted_average_students:
            print(
                f"Кафедра: {item['department_name']}, "
                f"Среднее количество студентов:
{item['average_number_of_students']:.2f}"
            )
        print("\n")

    return sorted_average_students

def findDepartmentsStartWithKey(many_to_many, key, debug=True):
    """
    Запрос 3: Кафедры, начинающиеся на букву == key и их студенческие группы
    """
    department_to_groups = defaultdict(list)
    for group_name, department_name in many_to_many:
        department_to_groups[department_name].append(group_name)

    filtered_departments = {
        dept: groups
        for dept, groups in department_to_groups.items()
        if dept.startswith(key)
    }

    result = [
        {"department_name": dept, "student_groups": groups}
        for dept, groups in filtered_departments.items()
    ]

    if debug:
        print(f"Запрос 3: Кафедры, начинающиеся на '{key}', и их студенческие
группы:")
        if not result:
            print("Ничего не найдено")
        else:
            for item in result:
                groups = "\n\t".join(item["student_groups"]) if
item["student_groups"] else "Нет групп"
                print(f"Кафедра {item['department_name']}:\n\t{groups}")
        print("\n")

    return result
```

StudentGroup.py

```
class StudentGroup:
    def __init__(self, id, group_name, number_of_students, department_id):
        self.id = id
        self.group_name = group_name
        self.number_of_students = number_of_students
        self.department_id = department_id

    def __repr__(self):
```

```
        return (f"StudentGroup(id={self.id}, group_name='{self.group_name}',
"
                f"number_of_students={self.number_of_students},
department_id={self.department_id})")
```

StudentGroupDepartment.py

```
class StudentGroupDepartment:
    def __init__(self, student_group_id, department_id):
        self.student_group_id = student_group_id
        self.department_id = department_id

    def __repr__(self):
        return
f"StudentGroupDepartment(student_group_id={self.student_group_id},
department_id={self.department_id})"
```

tests/data.py

```
from Department import Department
from StudentGroup import StudentGroup
from StudentGroupDepartment import StudentGroupDepartment

def get_departments():
    return [
        Department(id=1, name="Прикладная математика"),
        Department(id=2, name="Информатика"),
        Department(id=3, name="Физика"),
        Department(id=4, name="Астрономия"),
        Department(id=5, name="История"),
        Department(id=6, name="Археология"),
        Department(id=7, name="Астрофизика"),
    ]

def get_student_groups():
    return [
        StudentGroup(id=1, group_name="Группа 101", number_of_students=30,
department_id=1),
        StudentGroup(id=2, group_name="Группа 102", number_of_students=25,
department_id=2),
        StudentGroup(id=3, group_name="Группа 103", number_of_students=28,
department_id=1),
        StudentGroup(id=4, group_name="Группа 104", number_of_students=22,
department_id=3),
        StudentGroup(id=5, group_name="Группа 105", number_of_students=27,
department_id=4),
        StudentGroup(id=6, group_name="Группа 1003", number_of_students=20,
department_id=4),
        StudentGroup(id=7, group_name="Группа 104А", number_of_students=21,
department_id=5),
        StudentGroup(id=8, group_name="Группа 105А", number_of_students=24,
department_id=6),
        StudentGroup(id=9, group_name="Группа 1003А", number_of_students=23,
department_id=7),
        StudentGroup(id=10, group_name="Группа 104Б", number_of_students=21,
department_id=5),
        StudentGroup(id=11, group_name="Группа 145Б", number_of_students=20,
department_id=4),
        StudentGroup(id=12, group_name="Группа 1Б03", number_of_students=19,
department_id=7),
    ]
```

```
def get_student_group_departments():
    return [
        StudentGroupDepartment(student_group_id=1, department_id=1),
        StudentGroupDepartment(student_group_id=2, department_id=2),
        StudentGroupDepartment(student_group_id=3, department_id=1),
        StudentGroupDepartment(student_group_id=4, department_id=3),
        StudentGroupDepartment(student_group_id=5, department_id=4),
        StudentGroupDepartment(student_group_id=6, department_id=4),
        StudentGroupDepartment(student_group_id=7, department_id=5),
        StudentGroupDepartment(student_group_id=8, department_id=6),
        StudentGroupDepartment(student_group_id=9, department_id=7),
        StudentGroupDepartment(student_group_id=10, department_id=5),
        StudentGroupDepartment(student_group_id=11, department_id=4),
        StudentGroupDepartment(student_group_id=12, department_id=7),
    ]
```

tests/test_requests.py

```
import unittest

from Requests import findGroupsEndWithKey,
DepartmentsSortedByAvgNumberOfStudents, findDepartmentsStartWithKey


class TestRequests(unittest.TestCase):
    def setUp(self):
        # Подготовка тестовых данных
        self.one_to_many = [
            ("Группа 101", 30, "Прикладная математика"),
            ("Группа 102", 25, "Информатика"),
            ("Группа 103", 28, "Прикладная математика"),
            ("Группа 104", 22, "Физика"),
            ("Группа 105", 27, "Астрономия"),
            ("Группа 1003", 20, "Астрономия"),
            ("Группа 104А", 21, "История"),
            ("Группа 105А", 24, "Археология"),
            ("Группа 1003А", 23, "Астрофизика"),
            ("Группа 104Б", 21, "История"),
            ("Группа 145Б", 20, "Астрономия"),
            ("Группа 1Б03", 19, "Астрофизика"),
        ]

        self.many_to_many = [
            ("Группа 101", "Прикладная математика"),
            ("Группа 102", "Информатика"),
            ("Группа 103", "Прикладная математика"),
            ("Группа 104", "Физика"),
            ("Группа 105", "Астрономия"),
            ("Группа 1003", "Астрономия"),
            ("Группа 104А", "История"),
            ("Группа 105А", "Археология"),
            ("Группа 1003А", "Астрофизика"),
            ("Группа 104Б", "История"),
            ("Группа 145Б", "Астрономия"),
            ("Группа 1Б03", "Астрофизика"),
        ]

    def test_findGroupsEndWithKey_existing_key(self):
        # Проверка существующего ключа "03"
        result = findGroupsEndWithKey(self.one_to_many, "03", debug=False)
        expected = [
            {"group_name": "Группа 103", "department_name": "Прикладная
математика"},
```

```
            {"group_name": "Группа 1003", "department_name": "Астрономия"},
            {"group_name": "Группа 1Б03", "department_name": "Астрофизика"},
        ]
        self.assertEqual(result, expected)

    def test_DepartmentsSortedByAvgNumberOfStudents_correct_order(self):
        # Проверка сортировки кафедр по среднему количеству студентов
        result = DepartmentsSortedByAvgNumberOfStudents(self.one_to_many,
debug=False)
        expected = [
            {"department_name": "Прикладная математика",
"average_number_of_students": 29.0},
            {"department_name": "Информатика", "average_number_of_students":
25.0},
            {"department_name": "Астрономия", "average_number_of_students":
22.0},
            {"department_name": "Физика", "average_number_of_students":
22.0},
            {"department_name": "Археология", "average_number_of_students":
24.0},
            {"department_name": "Астрофизика", "average_number_of_students":
21.0},
            {"department_name": "История", "average_number_of_students":
20.5},
        ]
        # Прикладная математика: (30 + 28) / 2 = 29.0
        # Информатика: 25 / 1 = 25.0
        # Физика: 22 / 1 = 22.0
        # Астрономия: (27 + 20 + 20) / 3 ≈ 22.33
        # История: (21 + 21) / 2 = 21.0
        # Археология: 24 / 1 = 24.0
        # Астрофизика: (23 + 19) / 2 = 21.0

        expected_correct = [
            {"department_name": "Прикладная математика",
"average_number_of_students": 29.0},
            {"department_name": "Информатика", "average_number_of_students":
25.0},
            {"department_name": "Археология", "average_number_of_students":
24.0},
            {"department_name": "Астрономия", "average_number_of_students":
22.33},
            {"department_name": "Физика", "average_number_of_students":
22.0},
            {"department_name": "История", "average_number_of_students":
21.0},
            {"department_name": "Астрофизика", "average_number_of_students":
21.0},
        ]

        # Используем округление до 2 знаков для сравнения
        for res, exp in zip(result, expected_correct):
            self.assertEqual(res["department_name"], exp["department_name"])
            self.assertAlmostEqual(res["average_number_of_students"],
exp["average_number_of_students"], places=2)

    def test_findDepartmentsStartWithKey_no_matches(self):
        # Проверка ключа, для которого нет соответствий
        result = findDepartmentsStartWithKey(self.many_to_many, "Z",
debug=False)
        expected = []
        self.assertEqual(result, expected)

    def test_findDepartmentsStartWithKey_existing_key(self):
```

```
        # Дополнительный тест для существующего ключа "A"
        result = findDepartmentsStartWithKey(self.many_to_many, "A",
debug=False)
        expected = [
            {"department_name": "Астрономия", "student_groups": ["Группа
105", "Группа 1003", "Группа 145Б"]},
            {"department_name": "Археология", "student_groups": ["Группа
105А"]},
            {"department_name": "Астрофизика", "student_groups": ["Группа
1003А", "Группа 1Б03"]},
        ]
        self.assertEqual(result, expected)

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

Результат тестирования: