Main.py

```
from operator import itemgetter

from typing import List, Tuple, Dict



class Conductor:

    """Дирижер"""


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

        self.id = id

        self.name = name

        self.salary = salary



class Orchestra:

    """Оркестр"""


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

        self.id = id

        self.name = name

        self.conductor_id = conductor_id



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
```

```
def join_data(one: List[Conductor], many: List[Orchestra]) -> List[Tuple[str, int, str]]:
    """Соединение данных один-ко-многим"""
    return [(c.name, c.salary, o.name) for o in many for c in one if o.conductor_id == c.id]


def calculate_total_salary(orchestra: Orchestra, conductors: List[Conductor], conductor_orchestras:
List[ConductorOrchestra]) -> Tuple[str, int]:
    """Вычисление общей зарплаты для оркестра"""
    conductor_ids = [co.conductor_id for co in conductor_orchestras if co.orchestra_id ==
orchestra.id]
    total_salary = sum([c.salary for c in conductors if c.id in conductor_ids])
    return orchestra.name, total_salary


def find_conductors_for_symphony(orchestras: List[Orchestra], conductors: List[Conductor],
conductor_orchestras: List[ConductorOrchestra]) -> Dict[str, List[str]]:
    """Поиск дирижеров для Symphony Orchestra"""
    res_A3 = {}
    for orchestra in orchestras:
        if 'Symphony' in orchestra.name:
            conductor_ids = [co.conductor_id for co in conductor_orchestras if
co.orchestra_id == orchestra.id]
            conductor_names = [c.name for c in conductors if c.id in conductor_ids]
            res_A3[orchestra.name] = conductor_names
    return res_A3


if __name__ == '__main__':
    conductors = [
        Conductor(1, 'John Smith', 5000),
        Conductor(2, 'Emily Johnson', 6000),
        Conductor(3, 'Michael Davis', 5500)
    ]
```

```
	orchestras = [
		Orchestra(1, 'Symphony Orchestra', 1),
		Orchestra(2, 'Chamber Orchestra', 2),
		Orchestra(3, 'Philharmonic Orchestra', 3)
	]

	conductor_orchestras = [
		ConductorOrchestra(1, 1),
		ConductorOrchestra(2, 2),
		ConductorOrchestra(3, 3),
		ConductorOrchestra(1, 2),
		ConductorOrchestra(2, 1),
		ConductorOrchestra(3, 2),
	]

	# Задание A1
	res_A1 = join_data(conductors, orchestras)
	print('Задание A1')
	print(sorted(res_A1, key=itemgetter(2)))

	# Задание A2
	res_A2_unsorted = [calculate_total_salary(orchestra, conductors, conductor_orchestras) for
orchestra in orchestras]
	res_A2 = sorted(res_A2_unsorted, key=itemgetter(1), reverse=True)  # Сортировка по
убыванию суммарной зарплаты
	print('\nЗадание A2')
	print(res_A2)

	# Задание A3
	res_A3 = find_conductors_for_symphony(orchestras, conductors, conductor_orchestras)
	print('\nЗадание A3')
	print(res_A3)
```

test.py

```
import unittest
from main import *


class TestOrchestraProgram(unittest.TestCase):
    def setUp(self):
        self.conductors = [
            Conductor(1, 'John Smith', 5000),
            Conductor(2, 'Emily Johnson', 6000),
            Conductor(3, 'Michael Davis', 5500)
        ]

        self.orchestras = [
            Orchestra(1, 'Symphony Orchestra', 1),
            Orchestra(2, 'Chamber Orchestra', 2),
            Orchestra(3, 'Philharmonic Orchestra', 3)
        ]

        self.conductor_orchestras = [
            ConductorOrchestra(1, 1),
            ConductorOrchestra(2, 2),
            ConductorOrchestra(3, 3),
            ConductorOrchestra(1, 2),
            ConductorOrchestra(2, 1),
            ConductorOrchestra(3, 2),
        ]

    def test_join_data(self):
        result = join_data(self.conductors, self.orchestras)
        expected_result = [
            ('John Smith', 5000, 'Symphony Orchestra'),
            ('Emily Johnson', 6000, 'Chamber Orchestra'),
            ('Michael Davis', 5500, 'Philharmonic Orchestra')
        ]
        self.assertEqual(result, expected_result)

    def test_calculate_total_salary(self):
        orchestra = self.orchestras[0]  # Symphony Orchestra
        result = calculate_total_salary(orchestra, self.conductors, self.conductor_orchestras)
        expected_result = ('Symphony Orchestra', 11000)
        self.assertEqual(result, expected_result)

    def test_find_conductors_for_symphony(self):
        result = find_conductors_for_symphony(self.orchestras, self.conductors,
self.conductor_orchestras)
        expected_result = {'Symphony Orchestra': ['John Smith', 'Emily Johnson']}
        self.assertEqual(result, expected_result)


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