Текст программы

main.py

```
# используется для сортировки
from operator import itemgetter


class Emp:
    """Микропроцессор"""

    def __init__(self, id, name, cores, dep_id):
        self.id = id
        self.name = name
        self.cores = cores
        self.dep_id = dep_id


class Dep:
    """Компьютер"""

    def __init__(self, id, model):
        self.id = id
        self.model = model


class EmpDep:
    """
    'Микропроцессоры компьютера' для реализации
    связи многие-ко-многим
    """

    def __init__(self, dep_id, emp_id):
        self.dep_id = dep_id
        self.emp_id = emp_id


# Компьютеры
deps = [
    Dep(1, 'MacBook Air'),
    Dep(2, 'MacBook Pro'),
    Dep(3, 'iMac 2013'),

    Dep(11, 'ASUS 2000'),
    Dep(22, 'Acer 2003'),
    Dep(33, 'Honor 2020'),
]

# Микропроцессоры
emps = [
    Emp(1, 'Pentium', 2, 11),
    Emp(2, 'Celeron', 1, 11),
    Emp(3, 'Core i3', 4, 33),
    Emp(4, 'Core i7', 2, 33),
    Emp(5, 'M1', 8, 2),
]

emps_deps = [
    EmpDep(1, 1),
```

```
    EmpDep(1, 2),
    EmpDep(1, 3),
    EmpDep(3, 4),
    EmpDep(2, 5),

    EmpDep(11, 1),
    EmpDep(22, 2),
    EmpDep(33, 3),
    EmpDep(33, 4),
    EmpDep(33, 5),
]


def a1_solution(one_to_many):
    res_a1 = sorted(one_to_many, key=itemgetter(2))
    return res_a1


def a2_solution(one_to_many):
    res_a2_unsorted = []
    # Перебираем все компьютеры
    for d in deps:
        # Список микропроцессоров компьютера
        d_emps = list(filter(lambda i: i[2] == d.model, one_to_many))
        # Если в компьютере есть микропроцессоры
        if len(d_emps) > 0:
            # Кол-во ядер микропроцессоров компьютера
            d_sals = [sal for _, sal, _ in d_emps]
            # Сумма ядер микропроцессоров компьютера
            d_sals_sum = sum(d_sals)
            res_a2_unsorted.append((d.model, d_sals_sum))

    # Сортировка по сумме ядер
    res_a2 = sorted(res_a2_unsorted, key=itemgetter(1), reverse=True)
    return res_a2


def a3_solution(many_to_many):
    res_a3 = {}
    # Перебираем все компьютеры
    for d in deps:
        if 'Mac' in d.model:
            # Список микропроцессоров компьютеров
            d_emps = list(filter(lambda i: i[2] == d.model, many_to_many))
            # Только название микропроцессора
            d_emps_names = [x for x, _, _ in d_emps]
            # Добавляем результат в словарь
            # ключ - компьютер, значение - список названий микропроцессоров
            res_a3[d.model] = d_emps_names
    return res_a3


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

    # Соединение данных один-ко-многим
    one_to_many = [(e.name, e.cores, d.model)
                   for d in deps
                   for e in emps
                   if e.dep_id == d.id]

    # Соединение данных многие-ко-многим
```

```
    many_to_many_temp = [(d.model, ed.dep_id, ed.emp_id)
                         for d in deps
                         for ed in emps_deps
                         if d.id == ed.dep_id]

    many_to_many = [(e.name, e.cores, dep_name)
                    for dep_name, dep_id, emp_id in many_to_many_temp
                    for e in emps if e.id == emp_id]

    print('Задание А1')
    print(a1_solution(one_to_many))

    print('\nЗадание А2')
    print(a2_solution(one_to_many))

    print('\nЗадание А3')
    print(a3_solution(many_to_many))


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

tddtests.py

```
import unittest
from main import *


class TestRK2(unittest.TestCase):
    # Компьютеры
    deps = [
        Dep(1, 'MacBook Air'),
        Dep(2, 'MacBook Pro'),
        Dep(3, 'iMac 2013'),

        Dep(11, 'ASUS 2000'),
        Dep(22, 'Acer 2003'),
        Dep(33, 'Honor 2020'),
    ]

    # Микропроцессоры
    emps = [
        Emp(1, 'Pentium', 2, 11),
        Emp(2, 'Celeron', 1, 11),
        Emp(3, 'Core i3', 4, 33),
        Emp(4, 'Core i7', 2, 33),
        Emp(5, 'M1', 8, 2),
    ]

    def test_A1(self):
        one_to_many = [(e.name, e.cores, d.model)
                       for d in deps
                       for e in emps
                       if e.dep_id == d.id]
        self.assertEqual(a1_solution(one_to_many),
                         [('Pentium', 2, 'ASUS 2000'), ('Celeron', 1, 'ASUS
2000'), ('Core i3', 4, 'Honor 2020'),
                          ('Core i7', 2, 'Honor 2020'), ('M1', 8, 'MacBook
Pro')])
```

```
    def test_A2(self):
        one_to_many = [(e.name, e.cores, d.model)
                       for d in deps
                       for e in emps
                       if e.dep_id == d.id]
        self.assertEqual(a2_solution(one_to_many),
                         [('MacBook Pro', 8), ('Honor 2020', 6), ('ASUS 2000',
3)])

    def test_A3(self):
        many_to_many_temp = [(d.model, ed.dep_id, ed.emp_id)
                             for d in deps
                             for ed in emps_deps
                             if d.id == ed.dep_id]

        many_to_many = [(e.name, e.cores, dep_name)
                        for dep_name, dep_id, emp_id in many_to_many_temp
                        for e in emps if e.id == emp_id]
        self.assertEqual(a3_solution(many_to_many),
                         {'MacBook Air': ['Pentium', 'Celeron', 'Core i3'],
'MacBook Pro': ['M1'],
                          'iMac 2013': ['Core i7']})


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

Результаты выполнения

Пример успешного прохождения тестов

Ran 3 tests in 0.002s

OK

Пример неудачного прохождения тестов

Ran 3 tests in 0.010s

FAILED (failures=1)

{'MacBook Air': ['Pentium', 'Celeron', 'Core i3'],
 'MacBook Pro': ['M1'],
 'iMac 2013': ['Core i5']} != {'MacBook Air': ['Pentium', 'Celeron', 'Core i3'],
 'MacBook Pro': ['M1'],
 'iMac 2013': ['Core i7']}