Main.py

```
from operator import itemgetter
class Oper:
    """Оператор"""

    def __init__(self, id, description, syntax, arg_amount, prog_lang_id):
        self.id = id
        self.description = description
        self.syntax = syntax
        self.arg_amount = arg_amount
        self.prog_lang_id = prog_lang_id


class Proglang:
    """Язык программирования"""

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


class OperProglang:
    """
    связь многие-ко-многим
    """

    def __init__(self, oper_id, prog_lang_id):
        self.oper_id = oper_id
        self.prog_lang_id = prog_lang_id


# Языки программирования
prog_langs = [
    Proglang(1, "C++"),
    Proglang(2, "C#"),
    Proglang(3, "Pascal"),
    Proglang(4, "Python"),
    Proglang(5, "Java"),
]

# Операторы
opers = [
    Oper(1, "Array index", "[]", 2, 1),
    Oper(2, "Increment", "++", 1, 1),
    Oper(3, "Equality", "==", 2, 2),
    Oper(4, "Null coalescing", "??", 2, 2),
    Oper(5, "Assignment", ":=", 2, 3),
    Oper(6, "Exponentiation", "**", 2, 4),
    Oper(7, "Ternary operator", "?:", 3, 5),
]

opers_prog_langs = [
    OperProglang(1, 1),
    OperProglang(1, 2),
    OperProglang(1, 3),
    OperProglang(1, 4),
    OperProglang(1, 5),

    OperProglang(2, 1),
    OperProglang(2, 2),
    OperProglang(2, 5),

    OperProglang(3, 1),
```

```
    OperProglang(3, 2),
    OperProglang(3, 4),
    OperProglang(3, 5),

    OperProglang(4, 2),

    OperProglang(5, 3),

    OperProglang(6, 4),

    OperProglang(7, 1),
    OperProglang(7, 2),
    OperProglang(7, 5),
]


def a1_sol(one_to_many):
    # сортировка по полю description
    res_1 = sorted(one_to_many, key=itemgetter(0))
    return res_1


def a2_sol(one_to_many):
    # сортировка по сумме аргументов в каждом операторе в языке
    res_2 = []
    for pl in prog_langs:
        pl_ops = list(filter(lambda i: i[3] == pl.name, one_to_many))
        if len(pl_ops) > 0:
            pl_arg_am = [item[2] for item in pl_ops]
            pl_arg_sum = sum(pl_arg_am)
            res_2.append((pl.name, pl_arg_sum))
            return sorted(res_2, key=itemgetter(1))


def a3_sol(many_to_many):
    # вывод всех операторов, содержащих в синтаксисе '=', и языков
программирования, в которых они используются
    res_3 = {}
    for op in opers:
        if op.syntax.find('=') != -1:
            ops_pl = list(filter(lambda i: i[1] == op.syntax, many_to_many))
            ops_pl_name = [item[3] for item in ops_pl]
            res_3[op.syntax] = ops_pl_name

    return res_3
def main():
    """Основная функция"""

    # Соединение данных один-ко-многим
    one_to_many = [(op.description, op.syntax, op.arg_amount, pl.name)
                   for pl in prog_langs
                   for op in opers
                   if op.prog_lang_id == pl.id]

    # Соединение данных многие-ко-многим
    many_to_many_temp = [(pl.name, op_pl.prog_lang_id, op_pl.oper_id)
                         for pl in prog_langs
                         for op_pl in opers_prog_langs
                         if pl.id == op_pl.prog_lang_id]

    many_to_many = [(op.description, op.syntax, op.arg_amount, pl_name)
                    for pl_name, pl_id, op_id in many_to_many_temp
                    for op in opers if op.id == op_id]
```

```
    print('Задание A1')
    print(a1_sol(one_to_many))

    print('\nЗадание A2')
    print(a2_sol(one_to_many))


    print('\nЗадание A3')
    print(a3_sol(many_to_many))


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

tddtest.py

```
import unittest

from main import *

class MyTestCase(unittest.TestCase):
    # Языки программирования
    prog_langs = [
        Proglang(1, "C++"),
        Proglang(2, "C#"),
        Proglang(3, "Pascal"),
        Proglang(4, "Python"),
        Proglang(5, "Java"),
    ]

    # Операторы
    opers = [
        Oper(1, "Array index", "[]", 2, 1),
        Oper(2, "Increment", "++", 1, 1),
        Oper(3, "Equality", "==", 2, 2),
        Oper(4, "Null coalescing", "??", 2, 2),
        Oper(5, "Assignment", ":=", 2, 3),
        Oper(6, "Exponentiation", "**", 2, 4),
        Oper(7, "Ternary operator", "?:", 3, 5),
    ]

    opers_prog_langs = [
        OperProglang(1, 1),
        OperProglang(1, 2),
        OperProglang(1, 3),
        OperProglang(1, 4),
        OperProglang(1, 5),

        OperProglang(2, 1),
        OperProglang(2, 2),
        OperProglang(2, 5),

        OperProglang(3, 1),
        OperProglang(3, 2),
        OperProglang(3, 4),
        OperProglang(3, 5),

        OperProglang(4, 2),

        OperProglang(5, 3),

        OperProglang(6, 4),

        OperProglang(7, 1),
```

```
        OperProglang(7, 2),
        OperProglang(7, 5),
    ]

    def test_a1(self):
        one_to_many = [(op.description, op.syntax, op.arg_amount, pl.name)
                       for pl in prog_langs
                       for op in opers
                       if op.prog_lang_id == pl.id]
        self.assertEqual(a1_sol(one_to_many),
                         [('Array index', '[]', 2, 'C++'), ('Assignment',
':=', 2, 'Pascal'),
                          ('Equality', '==', 2, 'C#'), ('Exponentiation',
'**', 2, 'Python'),
                          ('Increment', '++', 1, 'C++'), ('Null coalescing',
'??', 2, 'C#'),
                          ('Ternary operator', '?:', 3, 'Java')])
    def test_a2(self):
        one_to_many = [(op.description, op.syntax, op.arg_amount, pl.name)
                       for pl in prog_langs
                       for op in opers
                       if op.prog_lang_id == pl.id]
        self.assertEqual(a2_sol(one_to_many),
                         [('C++', 3)])

    def test_a3(self):
        many_to_many_temp = [(pl.name, op_pl.prog_lang_id, op_pl.oper_id)
                             for pl in prog_langs
                             for op_pl in opers_prog_langs
                             if pl.id == op_pl.prog_lang_id]

        many_to_many = [(op.description, op.syntax, op.arg_amount, pl_name)
                        for pl_name, pl_id, op_id in many_to_many_temp
                        for op in opers if op.id == op_id]
        self.assertEqual(a3 _sol(many_to_many),
                         {'==': ['C++', 'C#', 'Python', 'Java'], ':=':
['Pascal']})


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

Результат выполнения:

```
C:\Users\aleks\PycharmProjects\RK1\venv\Scripts\python.exe C:\Users\aleks\PycharmProjects\RK1\venv\tddtes.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK

Process finished with exit code 0
```