Файл test.py

```
import unittest
import main

PLs_test: list[main.PL] = [
    main.PL(1, 'Python'),
    main.PL(2, 'C++'),
    main.PL(3, 'C#'),
    main.PL(4, 'Java')
]
Syntaxes_test = [
    main.Syntax(1, 'do-while', 2, 0.5),
    main.Syntax(2, 'pointer', 2, 2.0),
    main.Syntax(3, 'interface', 3, 2.0),
    main.Syntax(4, 'for', 1, 1.5),
    main.Syntax(5, 'class', 4, 2.0)
]
PL_Syns_test = [
    main.PL_Syn(1, 1),
    main.PL_Syn(1, 4),
    main.PL_Syn(1, 5),
    main.PL_Syn(2, 1),
    main.PL_Syn(2, 2),
    main.PL_Syn(2, 4),
    main.PL_Syn(2, 5),
    main.PL_Syn(3, 1),
    main.PL_Syn(3, 3),
    main.PL_Syn(3, 4),
    main.PL_Syn(3, 5),
    main.PL_Syn(4, 1),
    main.PL_Syn(4, 4),
    main.PL_Syn(4, 5)
]


class MyTestCase(unittest.TestCase):
    request1_expected_result = [('interface', 'C#'),
                                ('do-while', 'C++'),
                                ('pointer', 'C++'),
                                ('class', 'Java'),
                                ('for', 'Python')]
    request2_expected_result = [('C++', 2.5),
                                ('C#', 2.0),
                                ('Java', 2.0),
                                ('Python', 1.5)]
    request3_expected_result = {'C++': ['do-while', 'pointer', 'for',
'class'],
                                'C#': ['do-while', 'interface', 'for',
'class']}

    def test_request1(self):
        result = main.request1(PLs_test, Syntaxes_test)
        self.assertEqual(self.request1_expected_result, result)

    def test_request2(self):
        result = main.request2(PLs_test, Syntaxes_test)
        self.assertEqual(self.request2_expected_result, result)

    def test_request3(self):
        result = main.request3(PLs_test, Syntaxes_test, PL_Syns_test)
        self.assertEqual(self.request3_expected_result, result)
```

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

Файл main.py

```
from operator import itemgetter


class PL:
    def __init__(self, pl_id, name):
        self.pl_id = pl_id
        self.name = name


class Syntax:
    def __init__(self, syn_id, name, pl_id, usefulness_grade=1.0):
        self.syn_id = syn_id
        self.name = name
        self.pl_id = pl_id
        self.usefulness_grade = usefulness_grade


class PL_Syn:
    def __init__(self, pl_id, syn_id):
        self.pl_id = pl_id
        self.syn_id = syn_id


def request1(PLs, Syntaxes):
    one_to_many = [(syn.name, pl.name)
                   for pl in PLs
                   for syn in Syntaxes
                   if (pl.pl_id == syn.pl_id)]
    result = sorted(one_to_many, key=itemgetter(1))
    return result
    #print(*result, sep="\n")


def request2(PLs, Syntaxes):
    one_to_many = [(syn.name, syn.usefulness_grade, pl.name)
                   for pl in PLs
                   for syn in Syntaxes
                   if (pl.pl_id == syn.pl_id)]
    result = list()
    for pl in PLs:
        syns = list(filter(lambda i: i[2] == pl.name, one_to_many))
        if len(syns) != 0:
            sum_ug = sum(ug for _, ug, _ in syns)
            result.append((pl.name, sum_ug))
    result = sorted(result, key=itemgetter(1), reverse=True)
    return result
    #print(*result, sep="\n")


def request3(PLs, Syntaxes, PL_Syns):
    many_to_many_temp = [(pl.name, p_s.syn_id)
                         for pl in PLs
                         for p_s in PL_Syns
                         if p_s.pl_id == pl.pl_id]
    many_to_many = [(pl_name, syn.name)
                    for pl_name, syn_id in many_to_many_temp
                    for syn in Syntaxes
```

```
                    if syn.syn_id == syn_id]

    result = dict()
    for pl in PLs:
        if "C" in pl.name:
            filtered = list(filter(lambda i: i[0] == pl.name, many_to_many))
            s_names = [i for _, i in filtered]
            result[pl.name] = s_names
    return result
    #print(result)


def main():
    pass
```