Группа: ИУ5-32Б
Студент: Волощук Александр

Рубежный контроль 2

Условия:

Рубежный контроль представляет собой разработку тестов на языке Python.

1) Проведите рефакторинг текста программы рубежного контроля №1 таким образом, чтобы он был пригоден для модульного тестирования.

2) Для текста программы рубежного контроля №1 создайте модульные тесты с применением TDD - фреймворка (3 теста).

Текст программы:

```
# main.py

class Computer:
    def __init__(self, id: int, comp_class_id: int, os: str, cpu_mhz:
float) -> None:
        self.id = id
        self.comp_class_id = comp_class_id
        self.os = os
        self.cpu_mhz = cpu_mhz


class ComputerClass:
    def __init__(self, id: int, name: str) -> None:
        self.id = id
        self.name = name


class CompCompClass:
    def __init__(self, comp_id: int, comp_class_id: int) -> None:
        self.comp_id = comp_id
        self.comp_class_id = comp_class_id


computers: list[Computer] = [
    Computer(1, 1, 'Windows 8.1', 2.7),
    Computer(2, 1, 'Windows 11', 3.4),
    Computer(3, 2, 'Windows 10', 3.4),
    Computer(4, 2, 'Debian', 2.2),
    Computer(5, 3, 'Linux Mint', 2.2),
    Computer(6, 3, 'Windows 7', 1.8),
    Computer(7, 3, 'Windows XP', 1.8),
]

computer_classes: list[ComputerClass] = [
    ComputerClass(1, '128'),
    ComputerClass(2, '212A'),
    ComputerClass(3, '301'),
]

comp_to_comp_class: list[CompCompClass] = [
    CompCompClass(1, 2),
    CompCompClass(1, 1),
    CompCompClass(2, 2),
    CompCompClass(3, 2),
    CompCompClass(4, 1),
    CompCompClass(5, 3),
    CompCompClass(5, 3),
    CompCompClass(6, 1),
    CompCompClass(6, 3),
    CompCompClass(7, 3),
]


def task1(cc_c_list) -> None:
    cc_c_list_sorted = [
        ((cc_c[1], cc_c[2]), cc_c[0])
```

```
        for cc_c in sorted(cc_c_list, key=lambda cc_c: -cc_c[2])
    ]
    return cc_c_list_sorted


def task2(cc_c_list) -> None:
    cc_comp_count = list(
        (
            (
                comp_class.name,
                len(tuple(filter(lambda c_cc: c_cc[0] == comp_class.name,
cc_c_list))),
            )
            for comp_class in computer_classes
        )
    )
    return cc_comp_count


def task3(cc_c_list) -> None:
    comp_win_cc_name = list(
        ((cc_c[1], cc_c[2]), cc_c[0])
        for cc_c in cc_c_list
        if cc_c[1].lower().startswith('win')
    )
    return comp_win_cc_name


def main() -> None:
    one_to_many = tuple(
        (comp_class.name, comp.os, comp.cpu_mhz)
        for comp in computers
        for comp_class in computer_classes
        if comp.comp_class_id == comp_class.id
    )

    many_to_many = tuple(
        (comp_class.name, comp.os, comp.cpu_mhz)
        for comp in computers
        for comp_class in computer_classes
        for link in comp_to_comp_class
        if comp.id == link.comp_id and comp_class.id ==
link.comp_class_id
    )

    print('Task1:\n\n{}\n'.format(task1(one_to_many)))
    print('Task2:\n\n{}\n'.format(task2(one_to_many)))
    print('Task3:\n\n{}\n'.format(task3(many_to_many)))


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

```
# unit_tests.py

import unittest
import main


class TestTaskFunctions(unittest.TestCase):

    def test_task1(self):
        in_data = (
            ('128', 'Windows 8.1', 2.7),
            ('128', 'Windows 11', 3.4),
            ('212A', 'Windows 10', 3.4),
            ('212A', 'Debian', 2.2),
            ('301', 'Linux Mint', 2.2),
            ('301', 'Windows 7', 1.8),
            ('301', 'Windows XP', 1.8),
        )
        correct_out = [
            (('Windows 11', 3.4), '128'),
            (('Windows 10', 3.4), '212A'),
            (('Windows 8.1', 2.7), '128'),
            (('Debian', 2.2), '212A'),
            (('Linux Mint', 2.2), '301'),
            (('Windows 7', 1.8), '301'),
            (('Windows XP', 1.8), '301'),
        ]
        out = main.task1(in_data)
        self.assertEqual(out, correct_out)

    def test_task2(self):
        in_data = (
            ('128', 'Windows 8.1', 2.7),
            ('128', 'Windows 11', 3.4),
            ('212A', 'Windows 10', 3.4),
            ('212A', 'Debian', 2.2),
            ('301', 'Linux Mint', 2.2),
            ('301', 'Windows 7', 1.8),
            ('301', 'Windows XP', 1.8),
        )
        correct_out = [('128', 2), ('212A', 2), ('301', 3)]
        out = main.task2(in_data)
        self.assertEqual(out, correct_out)

    def test_task3(self):
        in_data = (
            ('128', 'Windows 8.1', 2.7),
            ('212A', 'Windows 8.1', 2.7),
            ('212A', 'Windows 11', 3.4),
            ('212A', 'Windows 10', 3.4),
            ('128', 'Debian', 2.2),
            ('301', 'Linux Mint', 2.2),
            ('301', 'Linux Mint', 2.2),
            ('128', 'Windows 7', 1.8),
            ('301', 'Windows 7', 1.8),
            ('301', 'Windows XP', 1.8),
        )
        correct_out = [
```

```
            (('Windows 8.1', 2.7), '128'),
            (('Windows 8.1', 2.7), '212A'),
            (('Windows 11', 3.4), '212A'),
            (('Windows 10', 3.4), '212A'),
            (('Windows 7', 1.8), '128'),
            (('Windows 7', 1.8), '301'),
            (('Windows XP', 1.8), '301'),
        ]
        out = main.task3(in_data)
        self.assertEqual(out, correct_out)


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

Результаты выполнения:

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
```