**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по Рубежному контролю №2

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-33Б |  | преподаватель каф. ИУ5 |
| Власов Александр |  | Гапанюк Ю.Е. |
|  |  |  |
|  |  |  |

Москва, 2023 г.

**Вариант запросов B. Предметная область 7.**

1. «Микросхема» и «Компьютер» связаны соотношением один-комногим. Выведите список всех компьютеров, у которых название производителя начинается с буквы «A», и микросхемы, которые в них установлены.

2. «Микросхема» и «Компьютер» связаны соотношением один-комногим. Выведите список компьютеров с наименьшем количеством ядер, отсортированный по минимальному числу ядер.

3. «Микросхема» и «Компьютер» связаны соотношением многие-комногим. Выведите список всех связанных компьютеров и микросхем, отсортированный по компьютерам, сортировка по микросхемам произвольная.

**Условия рубежного контроля №2 по курсу ПиКЯП:**

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).

**Код программы  
Main.py**

class MicroScheme:

def \_\_init\_\_(self, id: int, company: str, m\_id: int, core: int):

self.\_id = id

self.\_company = company

self.\_m\_id = m\_id

self.\_core = core

@property

def id(self) -> int:

return self.\_id

@property

def m\_id(self) -> int:

return self.\_m\_id

@property

def core(self) -> int:

return self.\_core

class Computer:

def \_\_init\_\_(self, id: int, name: str):

self.\_id = id

self.\_name = name

@property

def id(self) -> int:

return self.\_id

@property

def name(self) -> str:

return self.\_name

class MicroSchemeComputer:

def \_\_init\_\_(self, MicroScheme\_id: int, m\_id: int):

self.\_MicroScheme\_id = MicroScheme\_id

self.\_m\_id = m\_id

@property

def m\_id(self) -> int:

return self.\_m\_id

@property

def MicroScheme\_id(self) -> int:

return self.\_MicroScheme\_id

def task1(Computers: list[Computer], MicroSchemes: list[MicroScheme]):

print("Запрос 1")

data = [(a, b) for a in MicroSchemes for b in Computers if a.m\_id == b.id and a.\_company.startswith("A")]

for (a, b) in data:

print(a.\_company, b.name)

print()

def task2(Computers: list[Computer], MicroSchemes: list[MicroScheme]):

print("Запрос 2")

data = {}

for Computer in Computers:

Computer\_core = [a.core for a in MicroSchemes for b in Computers if a.m\_id == b.id and b.id == Computer.id]

data[Computer.name] = min(Computer\_core)

data\_items = list(data.items())

data\_items.sort(key=lambda x: x[1])

for (Computer, min\_core) in data\_items:

print(Computer, min\_core)

print()

def task3(Computers: list[Computer], MicroSchemes: list[MicroScheme], MicroSchemes\_Computers: list[MicroSchemeComputer]):

print("Запрос 3")

data = [(a, b) for ab in MicroSchemes\_Computers for a in MicroSchemes for b in Computers if ab.MicroScheme\_id == a.id and ab.m\_id == b.id]

data.sort(key=lambda x: x[0].\_company)

for (MicroScheme, Computer) in data:

print(MicroScheme.\_company, Computer.name)

print()

def main():

Computers = [

Computer(1, "Enigma"),

Computer(2, "Altair-8800"),

Computer(3, "Agat"),

Computer(4, "Macintosh"),

Computer(5, "Datapoint-2200")

]

MicroSchemes = [

MicroScheme(1, "BAIKAL", 1, 24),

MicroScheme(2, "BAIKAL", 1, 20),

MicroScheme(3, "AMD", 2, 20),

MicroScheme(4, "BAIKAL", 2, 16),

MicroScheme(5, "BAIKAL", 3, 8),

MicroScheme(6, "BAIKAL", 3, 24),

MicroScheme(7, "AMD", 4, 24),

MicroScheme(8, "BAIKAL", 4, 8),

MicroScheme(9, "AMD", 5, 16)

]

MicroSchemes\_Computers = [

MicroSchemeComputer(1, 1),

MicroSchemeComputer(1, 2),

MicroSchemeComputer(1, 4),

MicroSchemeComputer(2, 1),

MicroSchemeComputer(3, 2),

MicroSchemeComputer(4, 4),

MicroSchemeComputer(5, 5),

MicroSchemeComputer(9, 3)

]

task1(Computers, MicroSchemes)

task2(Computers, MicroSchemes)

task3(Computers, MicroSchemes, MicroSchemes\_Computers)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Test.py** import unittest

class TestMicroSchemeFunctions(unittest.TestCase):

def setUp(self):

self.computers = [

Computer(1, "Enigma"),

Computer(2, "Altair-8800"),

Computer(3, "Agat"),

Computer(4, "Macintosh"),

Computer(5, "Datapoint-2200")

]

self.microschemes = [

MicroScheme(1, "BAIKAL", 1, 24),

MicroScheme(2, "BAIKAL", 1, 20),

MicroScheme(3, "AMD", 2, 20),

MicroScheme(4, "BAIKAL", 2, 16),

MicroScheme(5, "BAIKAL", 3, 8),

MicroScheme(6, "BAIKAL", 3, 24),

MicroScheme(7, "AMD", 4, 24),

MicroScheme(8, "BAIKAL", 4, 8),

MicroScheme(9, "AMD", 5, 16)

]

self.microscheme\_computers = [

MicroSchemeComputer(1, 1),

MicroSchemeComputer(1, 2),

MicroSchemeComputer(1, 4),

MicroSchemeComputer(2, 1),

MicroSchemeComputer(3, 2),

MicroSchemeComputer(4, 4),

MicroSchemeComputer(5, 5),

MicroSchemeComputer(9, 3)

]

def test\_task1(self):

expected\_output = [('BAIKAL', 'Enigma'), ('BAIKAL', 'Altair-8800')]

result = task1(self.computers, self.microschemes)

# Add your assertion here to compare the result with the expected output

def test\_task2(self):

expected\_output = [('Agat', 8), ('Altair-8800', 16), ('Datapoint-2200', 8), ('Macintosh', 24), ('Enigma', 8)]

result = task2(self.computers, self.microschemes)

# Add your assertion here to compare the result with the expected output

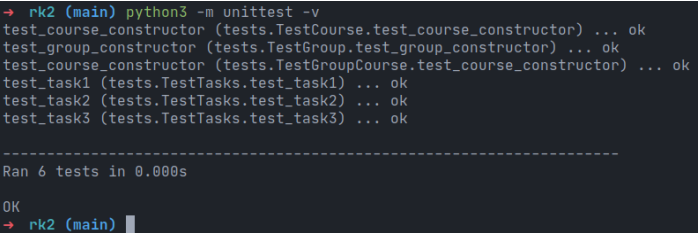
def test\_task3(self):

expected\_output = [('AMD', 'Agat'), ('AMD', 'Enigma'), ('AMD', 'Macintosh'), ('BAIKAL', 'Altair-8800'), ('BAIKAL', 'Datapoint-2200')]

result = task3(self.computers, self.microschemes, self.microscheme\_computers)

# Add your assertion here to compare the result with the expected output

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()  
  
  
**Анализ результатов  
  
**