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).

main_refactored.py

```
class Procedure:
    def __init__(self, id, name, size, db_id):
        self.id = id
        self.name = name
        self.size = size
        self.db_id = db_id


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


class ProcedureDB:
    def __init__(self, db_id, procedure_id):
        self.db_id = db_id
        self.procedure_id = procedure_id


def get_procedures_in_odd_databases(dbs, procedures):
    """Возвращает словарь с именами процедур для баз данных с нечетными номерами."""
    result = {}
    for db in dbs:
        if int(db.name[-1]) % 2 == 1:
```

```
        result[db.name] = [procedure.name for procedure in procedures if procedure.db_id ==
db.id]
    return result


def get_max_procedure_size_by_database(dbs, procedures):
    """Возвращает словарь с максимальными размерами процедур для каждой базы
данных."""
    result = {}
    for db in dbs:
        sizes = [procedure.size for procedure in procedures if procedure.db_id == db.id]
        if sizes:
            result[db.name] = max(sizes)
    return result


def get_procedures_by_connections(dbs, procedures, dbs_to_procedures):
    """Возвращает словарь с именами процедур для баз данных на основе связей."""
    result = {}
    for db in dbs:
        connected_procedures = []
        for connection in dbs_to_procedures:
            if db.id == connection.db_id:
                for procedure in procedures:
                    if procedure.id == connection.procedure_id:
                        connected_procedures.append(procedure.name)
        if connected_procedures:
            result[db.name] = connected_procedures
    return result


def main():
    dbs = [
```

```
    DataBase(1, 'db1'),
    DataBase(2, 'db2'),
    DataBase(3, 'db3')
]

procedures = [
    Procedure(1, 'p1', 16, 1),
    Procedure(2, 'p2', 32, 1),
    Procedure(3, 'p3', 64, 2),
    Procedure(4, 'p4', 128, 2),
    Procedure(5, 'p5', 256, 3),
    Procedure(6, 'p6', 512, 3)
]

dbs_to_procedures = [
    ProcedureDB(1, 1),
    ProcedureDB(1, 2),
    ProcedureDB(1, 3),
    ProcedureDB(2, 4),
    ProcedureDB(2, 5),
    ProcedureDB(2, 6),
    ProcedureDB(3, 1),
    ProcedureDB(3, 6)
]

print("Задание №1")
procedures_in_odd_dbs = get_procedures_in_odd_databases(dbs, procedures)
for db, procedure in procedures_in_odd_dbs.items():
    print(db, ':', ', '.join(procedure))
```

```
    print("\nЗадание №2")
    max_sizes = get_max_procedure_size_by_database(dbs, procedures)
    for db, size in sorted(max_sizes.items(), key=lambda x: x[1], reverse=True):
        print(db, ':', size)


    print("\nЗадание №3")
    procedures_by_connections = get_procedures_by_connections(dbs, procedures,
dbs_to_procedures)
    for db, procedure in procedures_by_connections.items():
        print(db, ':', ', '.join(procedure))


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

test_main.py

```
import unittest
from main_refactored import DataBase, Procedure, ProcedureDB,
get_procedures_in_odd_databases, get_max_procedure_size_by_database,
get_procedures_by_connections
class TestProcedures(unittest.TestCase):
    def setUp(self):
        self.dbs = [
            DataBase(1, 'db1'),
            DataBase(2, 'db2'),
            DataBase(3, 'db3')
        ]

        self.procedures = [
```

```
            Procedure(1, 'p1', 16, 1),
            Procedure(2, 'p2', 32, 1),
            Procedure(3, 'p3', 64, 2),
            Procedure(4, 'p4', 128, 2),
            Procedure(5, 'p5', 256, 3),
            Procedure(6, 'p6', 512, 3)
        ]

        self.dbs_to_procedures = [
            ProcedureDB(1, 1),
            ProcedureDB(1, 2),
            ProcedureDB(1, 3),
            ProcedureDB(2, 4),
            ProcedureDB(2, 5),
            ProcedureDB(2, 6),
            ProcedureDB(3, 1),
            ProcedureDB(3, 6)
        ]

    def test_get_procedures_in_odd_databases(self):
        result = get_procedures_in_odd_databases(self.dbs, self.procedures)
        expected = {'db1': ['p1', 'p2'], 'db3': ['p5', 'p6']}
        self.assertEqual(result, expected)

    def test_get_max_procedure_size_by_database(self):
        result = get_max_procedure_size_by_database(self.dbs, self.procedures)
        expected = {'db1': 32, 'db2': 128, 'db3': 512}
        self.assertEqual(result, expected)
```

```
    def test_get_procedures_by_connections(self):
        result = get_procedures_by_connections(self.dbs, self.procedures, self.dbs_to_procedures)
        expected = {
            'db1': ['p1', 'p2', 'p3'],
            'db2': ['p4', 'p5', 'p6'],
            'db3': ['p1', 'p6']
        }
        self.assertEqual(result, expected)


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