МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет “Информатика и системы управления”
Кафедра “Системы обработки информации и управления”

Дисциплина “Парадигмы и конструкции языков программирования”

Отчет по лабораторной работе №5
“ Модульное тестирование в Python”

Выполнил:
Студент группы ИУ5Ц-54Б
Цурин А.П.
Преподаватель:
Гапанюк Ю.Е.

Москва 2025

1. Задания для выполнения

Выберите любой фрагмент кода из лабораторных работ 1 или 2 или 3-4.

Модифицируйте код таким образом, чтобы он был пригоден для модульного тестирования.

Разработайте модульные тесты. В модульных тестах необходимо применить следующие технологии:

TDD - фреймворк (не менее 3 тестов).

BDD - фреймворк (не менее 3 тестов).

Создание Mock-объектов (необязательное дополнительное задание).

2. Листинг программы

Features/steps/math_operations_steps.py

```
"""
BDD шаги для тестирования математических операций
"""
from behave import given, when, then
from math_operations import field, filter_data, transform_data, calculate_average,
DataProcessor

@given('у меня есть список словарей')
def step_given_test_data(context):
    """Шаг: задание тестовых данных"""
    context.test_data = [
        {'name': 'Анна', 'age': 25, 'salary': 50000},
        {'name': 'Алексей', 'age': 30, 'salary': 60000},
        {'name': 'Мария', 'age': 28, 'salary': 55000},
        {'name': None, 'age': 35, 'salary': 70000}
    ]

@given('у меня есть DataProcessor с тестовыми данными')
def step_given_data_processor(context):
    """Шаг: создание DataProcessor"""
    test_data = [
        {'name': 'Анна', 'age': 25, 'salary': 50000},
        {'name': 'Алексей', 'age': 30, 'salary': 60000},
        {'name': 'Анна', 'age': 28, 'salary': 55000},  # Дубликат
        {'name': 'Мария', 'age': 35, 'salary': 70000}
    ]
    context.processor = DataProcessor(test_data)

@when('я выбираю поле "{field_name}" с помощью функции field')
def step_when_field_single(context, field_name):
    """Шаг: выборка одного поля"""
    context.result = list(field(context.test_data, field_name))

@when('я фильтрую данные по условию возраста больше {age:d}')
def step_when_filter_by_age(context, age):
    """Шаг: фильтрация по возрасту"""
    context.result = context.processor.filter_by_condition(lambda x: x['age'] > age)

@when('я получаю уникальные значения поля "{field_name}"')
```

```python
def step_when_unique_values(context, field_name):
    """Шаг: получение уникальных значений"""
    context.result = context.processor.get_unique_values(field_name)

@when('я вычисляю среднее значение поля "{field_name}"')
def step_when_calculate_average(context, field_name):
    """Шаг: вычисление среднего значения"""
    context.result = context.processor.calculate_field_average(field_name)

@then('я получаю список из {count:d} имен')
def step_then_get_names_list(context, count):
    """Шаг: проверка списка имен"""
    assert len(context.result) == count, f"Ожидалось {count} имен, получено
{len(context.result)}"

@then('я получаю {count:d} записей')
def step_then_get_records_count(context, count):
    """Шаг: проверка количества записей"""
    assert len(context.result) == count, f"Ожидалось {count} записей, получено
{len(context.result)}"

@then('я получаю {count:d} уникальных значений')
def step_then_get_unique_count(context, count):
    """Шаг: проверка количества уникальных значений"""
    assert len(context.result) == count, f"Ожидалось {count} уникальных значений,
получено {len(context.result)}"

@then('среднее значение равно {expected_value:f}')
def step_then_average_value(context, expected_value):
    """Шаг: проверка среднего значения"""
    assert abs(context.result - expected_value) < 0.01, f"Ожидалось {expected_value},
получено {context.result}"
```

features/math_operations.feature

```
# language: ru
Функционал: Математические операции для обработки данных

  Контекст: Как разработчик
    Я хочу иметь надежные функции для обработки данных
    Чтобы обеспечить корректность бизнес-логики

  Сценарий: Выборка поля из словарей
    Дано у меня есть список словарей
    Когда я выбираю поле "name" с помощью функции field
    Тогда я получаю список из 3 имен

  Сценарий: Фильтрация данных по условию
    Дано у меня есть DataProcessor с тестовыми данными
    Когда я фильтрую данные по условию возраста больше 28
    Тогда я получаю 2 записей

  Сценарий: Получение уникальных значений
    Дано у меня есть DataProcessor с тестовыми данными
    Когда я получаю уникальные значения поля "name"
```

Тогда я получаю 3 уникальных значений

Сценарий: Вычисление среднего значения
Дано у меня есть DataProcessor с тестовыми данными
Когда я вычисляю среднее значение поля "salary"
Тогда среднее значение равно 58750.0

Math_operations.py

```
"""
Модуль математических операций для тестирования
"""
import random

def field(items, *args):
    """
    Генератор для выборки полей из словарей
    """
    if len(args) == 0:
        return

    for item in items:
        if not isinstance(item, dict):
            continue

        if len(args) == 1:
            field_name = args[0]
            if field_name in item and item[field_name] is not None:
                yield item[field_name]
        else:
            result = {}
            has_valid_fields = False
            for field_name in args:
                if field_name in item and item[field_name] is not None:
                    result[field_name] = item[field_name]
                    has_valid_fields = True
            if has_valid_fields:
                yield result

def gen_random(num_count, min_value, max_value):
    """
    Генератор случайных чисел
    """
    return [random.randint(min_value, max_value) for _ in range(num_count)]

def filter_data(data, condition_func):
    """
    Фильтрация данных по условию
    """
    return list(filter(condition_func, data))

def transform_data(data, transform_func):
    """
    Преобразование данных с помощью функции
    """
```

```
    return list(map(transform_func, data))

def calculate_average(numbers):
    """
    Вычисление среднего значения списка чисел
    """
    if not numbers:
        return 0
    return sum(numbers) / len(numbers)

class DataProcessor:
    """Класс для обработки данных"""

    def __init__(self, data):
        self.data = data

    def get_unique_values(self, field_name):
        """Получить уникальные значения поля"""
        values = list(field(self.data, field_name))
        return list(set(values))

    def filter_by_condition(self, condition_func):
        """Отфильтровать данные по условию"""
        return filter_data(self.data, condition_func)

    def calculate_field_average(self, field_name):
        """Вычислить среднее значение поля"""
        values = list(field(self.data, field_name))
        # Фильтруем только числовые значения
        numeric_values = [v for v in values if isinstance(v, (int, float))]
        return calculate_average(numeric_values)
```

print_result.py

```
"""
Декоратор для вывода результатов
"""
from functools import wraps

def print_result(func):
    """
    Декоратор для вывода результата функции
    """
    @wraps(func)
    def wrapper(*args, **kwargs):
        result = func(*args, **kwargs)

        print(f"\nРезультат функции {func.__name__}:")
        if isinstance(result, dict):
            for key, value in result.items():
                print(f"  {key}: {value}")
        elif isinstance(result, list):
            for item in result:
                print(f"  {item}")
```

```
        else:
            print(f"  {result}")

        return result
    return wrapper
```

process_data.py

```
"""
Основной модуль обработки данных
"""
from math_operations import field, gen_random, filter_data, transform_data,
DataProcessor
from print_result import print_result

@print_result
def process_pipeline(data):
    """
    Основной пайплайн обработки данных
    """
    # Шаг 1: Получить уникальные имена
    unique_names = sorted(list(set(field(data, 'name'))), key=lambda x: x.lower())

    # Шаг 2: Отфильтровать имена, начинающиеся на 'A'
    filtered_names = filter_data(unique_names, lambda x: x.lower().startswith('a'))

    # Шаг 3: Добавить фразу к каждому имени
    transformed_names = transform_data(filtered_names, lambda x: f"{x} смотрит в
будущее")

    # Шаг 4: Сопоставить имена со случайными зарплатами
    result_dict = dict(zip(transformed_names, gen_random(len(transformed_names),
100000, 200000)))

    return result_dict

def main():
    """Основная функция"""
    # Тестовые данные
    test_data = [
        {'name': 'Анна', 'age': 25, 'salary': 50000},
        {'name': 'Алексей', 'age': 30, 'salary': 60000},
        {'name': 'Мария', 'age': 28, 'salary': 55000},
        {'name': 'Андрей', 'age': 35, 'salary': 70000},
        {'name': 'Ольга', 'age': 22, 'salary': 45000}
    ]

    # Обработка данных
    result = process_pipeline(test_data)
    return result

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

test_math_operations.py

```
"""
Модульные тесты с использованием TDD подхода (unittest)
"""
import unittest
from math_operations import field, gen_random, filter_data, transform_data,
calculate_average, DataProcessor

class TestMathOperationsTDD(unittest.TestCase):
    """TDD тесты для математических операций"""

    def setUp(self):
        """Настройка тестовых данных"""
        self.test_data = [
            {'name': 'Анна', 'age': 25, 'salary': 50000},
            {'name': 'Алексей', 'age': 30, 'salary': 60000},
            {'name': 'Мария', 'age': 28, 'salary': 55000},
            {'name': None, 'age': 35, 'salary': 70000},  # name = None
            {'name': 'Ольга', 'salary': 45000}  # отсутствует age
        ]

    def test_field_single_field(self):
        """Тест выборки одного поля"""
        # Act
        result = list(field(self.test_data, 'name'))

        # Assert - должны получить все не-None значения, включая Ольгу
        expected_names = ['Анна', 'Алексей', 'Мария', 'Ольга']
        self.assertEqual(result, expected_names)

    def test_field_multiple_fields(self):
        """Тест выборки нескольких полей"""
        # Act
        result = list(field(self.test_data, 'name', 'age'))

        # Assert - должны получить словари для всех элементов, где есть хотя бы одно
поле
        self.assertEqual(len(result), 5)  # Все 5 элементов, но с разными наборами
полей

        # Проверяем конкретные случаи
        for item in result:
            self.assertIsInstance(item, dict)

    def test_field_multiple_fields_specific(self):
        """Тест выборки нескольких полей с проверкой конкретных значений"""
        # Arrange
        data = [
            {'name': 'Анна', 'age': 25},
            {'name': 'Борис'},  # нет age
            {'age': 30},  # нет name
            {'name': None, 'age': 35}  # name = None
        ]
```

```
        # Act
        result = list(field(data, 'name', 'age'))

        # Assert
        self.assertEqual(len(result), 4)

        # Проверяем содержимое
        expected_results = [
            {'name': 'Анна', 'age': 25},
            {'name': 'Борис'},
            {'age': 30},
            {'age': 35}  # name = None не включается в результат для множественных
полей
        ]

        for i, expected in enumerate(expected_results):
            self.assertEqual(result[i], expected)

    def test_filter_data(self):
        """Тест фильтрации данных"""
        # Arrange
        data = [1, 2, 3, 4, 5, 6]

        # Act
        result = filter_data(data, lambda x: x % 2 == 0)

        # Assert
        self.assertEqual(result, [2, 4, 6])

    def test_transform_data(self):
        """Тест преобразования данных"""
        # Arrange
        data = [1, 2, 3]

        # Act
        result = transform_data(data, lambda x: x * 2)

        # Assert
        self.assertEqual(result, [2, 4, 6])

    def test_calculate_average(self):
        """Тест вычисления среднего значения"""
        # Arrange
        numbers = [10, 20, 30, 40]

        # Act
        result = calculate_average(numbers)

        # Assert
        self.assertEqual(result, 25.0)

    def test_calculate_average_empty_list(self):
```

```python
        """Тест вычисления среднего для пустого списка"""
        # Act
        result = calculate_average([])

        # Assert
        self.assertEqual(result, 0)

class TestDataProcessorTDD(unittest.TestCase):
    """TDD тесты для DataProcessor"""

    def setUp(self):
        """Настройка тестовых данных"""
        self.test_data = [
            {'name': 'Анна', 'age': 25, 'salary': 50000},
            {'name': 'Алексей', 'age': 30, 'salary': 60000},
            {'name': 'Анна', 'age': 28, 'salary': 55000},  # Дубликат имени
            {'name': 'Мария', 'age': 35, 'salary': 70000},
            {'name': 'Ольга', 'age': 22},  # Нет salary
            {'name': None, 'age': 40, 'salary': 80000},  # name = None
            {'age': 45, 'salary': 90000}  # Нет name
        ]
        self.processor = DataProcessor(self.test_data)

    def test_get_unique_values(self):
        """Тест получения уникальных значений"""
        # Act
        result = self.processor.get_unique_values('name')

        # Assert - должны получить 4 уникальных имени: Анна, Алексей, Мария, Ольга
        # None и отсутствующее имя не включаются
        self.assertEqual(len(result), 4)
        self.assertIn('Анна', result)
        self.assertIn('Алексей', result)
        self.assertIn('Мария', result)
        self.assertIn('Ольга', result)

    def test_get_unique_values_age(self):
        """Тест получения уникальных значений для поля age"""
        # Act
        result = self.processor.get_unique_values('age')

        # Assert - все 7 элементов имеют age
        self.assertEqual(len(result), 7)

    def test_filter_by_condition(self):
        """Тест фильтрации по условию"""
        # Act
        result = self.processor.filter_by_condition(lambda x: x.get('age', 0) > 28)

        # Assert - 4 элемента с age > 28
        self.assertEqual(len(result), 4)

    def test_calculate_field_average(self):
```

```
        """Тест вычисления среднего значения поля"""
        # Act
        result = self.processor.calculate_field_average('salary')

        # Assert - считаем среднее только для числовых значений salary
        # Элементы с salary: 50000, 60000, 55000, 70000, 80000, 90000
        salaries = [50000, 60000, 55000, 70000, 80000, 90000]
        expected_avg = sum(salaries) / len(salaries)
        self.assertAlmostEqual(result, expected_avg)

    def test_calculate_field_average_mixed_data(self):
        """Тест вычисления среднего для поля со смешанными типами"""
        # Arrange
        mixed_data = [
            {'value': 10},
            {'value': 'string'},  # Не число - игнорируется
            {'value': 20},
            {'value': None},  # None - игнорируется
            {'value': 30}
        ]
        processor = DataProcessor(mixed_data)

        # Act
        result = processor.calculate_field_average('value')

        # Assert - только числовые значения: 10, 20, 30
        expected_avg = (10 + 20 + 30) / 3
        self.assertAlmostEqual(result, expected_avg)

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

test_mock_operations.py

```
"""
Тесты с использованием Mock объектов
"""
import unittest
from unittest.mock import patch, MagicMock
from math_operations import gen_random, DataProcessor

class TestMockOperations(unittest.TestCase):
    """Тесты с Mock объектами"""

    @patch('math_operations.random.randint')
    def test_gen_random_with_mock(self, mock_randint):
        """Тест gen_random с mock random.randint"""
        # Arrange
        mock_randint.return_value = 42

        # Act
        result = gen_random(5, 1, 100)

        # Assert
```

```
        self.assertEqual(result, [42, 42, 42, 42, 42])
        self.assertEqual(mock_randint.call_count, 5)
        # Проверяем, что mock вызывался с правильными аргументами
        mock_randint.assert_called_with(1, 100)

    def test_data_processor_with_mock_data(self):
        """Тест DataProcessor с mock данными"""
        # Arrange
        mock_data = [
            {'name': 'Test1', 'age': 25, 'salary': 100},
            {'name': 'Test2', 'age': 30, 'salary': 200},
            {'name': 'Test1', 'age': 35, 'salary': 300}  # Дубликат имени
        ]
        processor = DataProcessor(mock_data)

        # Act & Assert
        unique_names = processor.get_unique_values('name')
        self.assertEqual(len(unique_names), 2)
        self.assertIn('Test1', unique_names)
        self.assertIn('Test2', unique_names)

        filtered = processor.filter_by_condition(lambda x: x['age'] > 25)
        self.assertEqual(len(filtered), 2)

        average_salary = processor.calculate_field_average('salary')
        self.assertEqual(average_salary, 200.0)

    @patch('math_operations.field')
    def test_calculate_average_with_mock_field(self, mock_field):
        """Тест calculate_field_average с mock field"""
        # Arrange
        mock_field.return_value = [100, 200, 300, 400]  # Mock возвращает значения
зарплат
        processor = DataProcessor([{'name': 'Test', 'salary': 100}])  # Любые данные

        # Act
        result = processor.calculate_field_average('salary')

        # Assert
        self.assertEqual(result, 250.0)  # (100+200+300+400)/4 = 250
        mock_field.assert_called_once_with([{'name': 'Test', 'salary': 100}], 'salary')

    @patch('math_operations.gen_random')
    def test_process_pipeline_with_mock_gen_random(self, mock_gen_random):
        """Тест пайплайна обработки с mock gen_random"""
        # Arrange
        mock_gen_random.return_value = [150000, 180000]

        # Импортируем здесь, чтобы mock применился
        from process_data import process_pipeline

        test_data = [
            {'name': 'Анна', 'salary': 50000},
```

```
            {'name': 'Андрей', 'salary': 60000},
            {'name': 'Мария', 'salary': 55000}
        ]

        # Act
        result = process_pipeline(test_data)

        # Assert
        self.assertIsInstance(result, dict)
        self.assertEqual(len(result), 2)
        mock_gen_random.assert_called_once_with(2, 100000, 200000)

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

3. Результаты работы программы

```
=== ЗАПУСК ВСЕХ ТЕСТОВ ===
Запуск TDD тестов...
test_calculate_field_average (test_math_operations_tdd.TestDataProcessorTDD.test_calculate_field_average)
Тест вычисления среднего значения поля ... ok
test_calculate_field_average_mixed_data
(test_math_operations_tdd.TestDataProcessorTDD.test_calculate_field_average_mixed_data)
Тест вычисления среднего для поля со смешанными типами ... ok
test_filter_by_condition (test_math_operations_tdd.TestDataProcessorTDD.test_filter_by_condition)
Тест фильтрации по условию ... ok
test_get_unique_values (test_math_operations_tdd.TestDataProcessorTDD.test_get_unique_values)
Тест получения уникальных значений ... ok
test_get_unique_values_age (test_math_operations_tdd.TestDataProcessorTDD.test_get_unique_values_age)
Тест получения уникальных значений для поля age ... ok
test_calculate_average (test_math_operations_tdd.TestMathOperationsTDD.test_calculate_average)
Тест вычисления среднего значения ... ok
test_calculate_average_empty_list (test_math_operations_tdd.TestMathOperationsTDD.test_calculate_average_empty_list)
Тест вычисления среднего для пустого списка ... ok
test_field_multiple_fields (test_math_operations_tdd.TestMathOperationsTDD.test_field_multiple_fields)
Тест выборки нескольких полей ... ok
test_field_multiple_fields_specific (test_math_operations_tdd.TestMathOperationsTDD.test_field_multiple_fields_specific)
Тест выборки нескольких полей с проверкой конкретных значений ... ok
test_field_single_field (test_math_operations_tdd.TestMathOperationsTDD.test_field_single_field)
Тест выборки одного поля ... ok
test_filter_data (test_math_operations_tdd.TestMathOperationsTDD.test_filter_data)
Тест фильтрации данных ... ok
test_transform_data (test_math_operations_tdd.TestMathOperationsTDD.test_transform_data)
Тест преобразования данных ... ok

----------------------------------------------------------------------
Ran 12 tests in 0.004s

OK

Запуск BDD тестов...
USING RUNNER: behave.runner:Runner
Функционал: Математические операции для обработки данных # features/math_operations.feature:2

  Функционал: Математические операции для обработки данных  # features/math_operations.feature:2

  Сценарий: Выборка поля из словарей                  # features/math_operations.feature:8
    Дано у меня есть список словарей                  # features/steps/math_operations_steps.py:7
    Когда я выбираю поле "name" с помощью функции field # features/steps/math_operations_steps.py:28
    Тогда я получаю список из 3 имен                  # features/steps/math_operations_steps.py:48
```

```
  Сценарий: Фильтрация данных по условию                  # features/math_operations.feature:13
    Дано у меня есть DataProcessor с тестовыми данными    # features/steps/math_operations_steps.py:17
    Когда я фильтрую данные по условию возраста больше 28 # features/steps/math_operations_steps.py:33
    Тогда я получаю 2 записей                             # features/steps/math_operations_steps.py:53

  Сценарий: Получение уникальных значений                 # features/math_operations.feature:18
    Дано у меня есть DataProcessor с тестовыми данными    # features/steps/math_operations_steps.py:17
    Когда я получаю уникальные значения поля "name"       # features/steps/math_operations_steps.py:38
    Тогда я получаю 3 уникальных значений                 # features/steps/math_operations_steps.py:58

  Сценарий: Вычисление среднего значения                  # features/math_operations.feature:23
    Дано у меня есть DataProcessor с тестовыми данными    # features/steps/math_operations_steps.py:17
    Когда я вычисляю среднее значение поля "salary"       # features/steps/math_operations_steps.py:43
    Тогда среднее значение равно 58750.0                  # features/steps/math_operations_steps.py:63

1 feature passed, 0 failed, 0 skipped
4 scenarios passed, 0 failed, 0 skipped
12 steps passed, 0 failed, 0 skipped
Took 0min 0.004s


Запуск Mock тестов...
test_calculate_average_with_mock_field
(test_mock_operations.TestMockOperations.test_calculate_average_with_mock_field)
Тест calculate_field_average с mock field ... ok
test_data_processor_with_mock_data (test_mock_operations.TestMockOperations.test_data_processor_with_mock_data)
Тест DataProcessor с mock данными ... ok
test_gen_random_with_mock (test_mock_operations.TestMockOperations.test_gen_random_with_mock)
Тест gen_random с mock random.randint ... ok
test_process_pipeline_with_mock_gen_random
(test_mock_operations.TestMockOperations.test_process_pipeline_with_mock_gen_random)
Тест пайплайна обработки с mock gen_random ...
Результат функции process_pipeline:
  Андрей смотрит в будущее: 150000
  Анна смотрит в будущее: 180000
ok

----------------------------------------------------------------------
Ran 4 tests in 0.006s

OK

=== РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ===
TDD тесты: ПРОЙДЕНЫ
BDD тесты: ПРОЙДЕНЫ
Mock тесты: ПРОЙДЕНЫ

 Все тесты пройдены успешно!
```