Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЁТ

по лабораторной работе №2.22

Дисциплина: «Анализ данных»
Тема: «Тестирование в Python [unittest]»

Выполнил:
Епифанов Алексей Александрович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной
техники и автоматизированных систем
», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Цель: приобретение навыков написания автоматизированных тестов на языке программирования Python версии 3.x.

Порядок выполнения работы:

1. Создал новый репозиторий, клонировал его, в нем создал ветку developer и перешел на нее.

2. Выполнил индивидуальное задание: Для индивидуального задания лабораторной работы 2.21 добавьте тесты с использованием модуля unittest, проверяющие операции по работе с базой данных.

```
import sqlite3
import unittest
from pathlib import Path

from program.ind import add_route, create_db, select_all, select_routes


class TestCreateDB(unittest.TestCase):
    def setUp(self):
        self.test_dir = Path("test.db")

    def tearDown(self):
        self.test_dir.unlink(missing_ok=True)

    def test_create_database(self):
        TestRoutes.not_skip = False
        create_db(self.test_dir)
        self.assertTrue(self.test_dir.exists())

        conn = sqlite3.connect(self.test_dir)
        cursor = conn.cursor()

        # Проверка наличия таблицы 'start'
        cursor.execute(
            "SELECT name FROM sqlite_master "
            "WHERE type='table' AND name='start';"
        )
        self.assertTrue(cursor.fetchone())

        # Проверка структуры таблицы 'start'
        cursor.execute("PRAGMA table_info(start);")
        columns = cursor.fetchall()
        expected_columns = [
```

```
            (0, "start_id", "INTEGER", 0, None, 1),
            (1, "start_point", "TEXT", 1, None, 0),
        ]
        self.assertEqual(columns, expected_columns)

        # Проверка наличия таблицы 'routes'
        cursor.execute(
            "SELECT name FROM sqlite_master "
            "WHERE type='table' AND name='routes';"
        )
        self.assertTrue(cursor.fetchone())

        # Проверка структуры таблицы 'routes'
        cursor.execute("PRAGMA table_info(routes);")
        columns = cursor.fetchall()
        expected_columns = [
            (0, "start_id", "INTEGER", 1, None, 0),
            (1, "route_number", "INTEGER", 0, None, 1),
            (2, "end_point", "TEXT", 1, None, 0),
        ]
        self.assertEqual(columns, expected_columns)

        conn.close()

        TestRoutes.not_skip = True


class TestRoutes(unittest.TestCase):
    not_skip = True

    @classmethod
    def setUpClass(self):
        # Пропускаем этот класс, если предыдущий класс
        # тестирования завершился неудачно
        if self.not_skip:
            pass
        else:
            raise unittest.SkipTest("create_db fail")

    def setUp(self):
        self.db_path = Path("test.db")
        create_db(self.db_path)

    def tearDown(self):
        self.db_path.unlink(missing_ok=True)
```

```
def test_add_route(self):
    add_route(self.db_path, "A", "B", 1)
    add_route(self.db_path, "C", "D", 2)

    conn = sqlite3.connect(self.db_path)
    cursor = conn.cursor()

    cursor.execute("SELECT COUNT(*) FROM start")
    self.assertEqual(cursor.fetchone()[0], 2)

    cursor.execute("SELECT COUNT(*) FROM routes")
    self.assertEqual(cursor.fetchone()[0], 2)

    cursor.execute(
        """
        SELECT start.start_point, routes.end_point, routes.route_number
        FROM start
        INNER JOIN routes ON routes.start_id = start.start_id
        """
    )
    rows = cursor.fetchall()
    self.assertEqual(len(rows), 2)
    self.assertDictEqual(
        {
            "начальный пункт": rows[0][0],
            "конечный пункт": rows[0][1],
            "номер маршрута": rows[0][2],
        },
        {
            "начальный пункт": "a",
            "конечный пункт": "b",
            "номер маршрута": 1,
        },
    )
    self.assertDictEqual(
        {
            "начальный пункт": rows[1][0],
            "конечный пункт": rows[1][1],
            "номер маршрута": rows[1][2],
        },
        {
            "начальный пункт": "c",
            "конечный пункт": "d",
            "номер маршрута": 2,
```

```
            },
        )

        conn.close()

    def test_select_all(self):
        add_route(self.db_path, "A", "B", 1)
        add_route(self.db_path, "C", "D", 2)

        routes = select_all(self.db_path)
        self.assertEqual(len(routes), 2)
        self.assertDictEqual(
            routes[0],
            {
                "начальный пункт": "a",
                "конечный пункт": "b",
                "номер маршрута": 1,
            },
        )
        self.assertDictEqual(
            routes[1],
            {
                "начальный пункт": "c",
                "конечный пункт": "d",
                "номер маршрута": 2,
            },
        )

    def test_select_routes(self):
        add_route(self.db_path, "A", "B", 1)
        add_route(self.db_path, "C", "D", 2)
        add_route(self.db_path, "B", "C", 3)

        routes = select_routes(self.db_path, "B")
        self.assertEqual(len(routes), 2)
        self.assertDictEqual(
            routes[0],
            {
                "начальный пункт": "a",
                "конечный пункт": "b",
                "номер маршрута": 1,
            },
        )
        self.assertDictEqual(
            routes[1],
```

```
        {
            "начальный пункт": "b",
            "конечный пункт": "c",
            "номер маршрута": 3,
        },
    )


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

```
❯ p_test tests -v
test_create_database (program.tests.TestCreateDB.test_create_database) ... ok
test_add_route (program.tests.TestRoutes.test_add_route) ... ok
test_select_all (program.tests.TestRoutes.test_select_all) ... ok
test_select_routes (program.tests.TestRoutes.test_select_routes) ... ok

----------------------------------------------------------------------
Ran 4 tests in 0.036s

OK
```

Рисунок 1. Результат запуска всех тестов

```
❯ p_test tests.TestCreateDB -v
test_create_database (program.tests.TestCreateDB.test_create_database) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.005s

OK
❯ p_test tests.TestRoutes -v
test_add_route (program.tests.TestRoutes.test_add_route) ... ok
test_select_all (program.tests.TestRoutes.test_select_all) ... ok
test_select_routes (program.tests.TestRoutes.test_select_routes) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.028s

OK
❯ p_test tests.TestRoutes.test_select_routes -v
test_select_routes (program.tests.TestRoutes.test_select_routes) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.011s

OK
 ~/Desktop/./../gits/Lab_2.22  lab-2-22-py3.12
```

Рисунок 2. Запуск отдельных тестов

Контрольные вопросы:

1. Для чего используется автономное тестирование?

Автономное тестирование используется для автоматической проверки корректности работы программного обеспечения без вмешательства человека. Это позволяет обнаруживать ошибки, повышать качество кода и ускорять процесс разработки.

2. Какие фреймворки Python получили наибольшее распространение для решения задач автономного тестирования?

Фреймворки Python для автономного тестирования: наибольшее распространение получили unittest (входит в стандартную библиотеку Python), pytest и nose (несмотря на то что разработка nose остановлена, его форк nose2 продолжает развиваться).

3. Какие существуют основные структурные единицы модуля unittest?

Основные структурные единицы модуля unittest:

TestCase: класс, представляющий отдельный тестовый случай.

TestSuite: коллекция тестовых случаев или тестовых наборов.

TestLoader: для загрузки тестов из TestCase и TestSuite.

TextTestRunner: класс для выполнения тестов и вывода результатов в текстовой форме.

TestResult: хранит результаты тестов.

4. Какие существуют способы запуска тестов unittest?

Способы запуска тестов unittest:

Использование интерфейса командной строки для запуска тестов (например, python -m unittest discover).

Использование unittest.main() внутри скрипта для запуска тестов.

Создание и использование объекта TestSuite для более сложных сценариев запуска.

5. Каково назначение класса TestCase?

Назначение класса TestCase: предоставляет рамки для создания тестовых случаев, включая методы для подготовки перед тестами (например,

setUp) и очистки после тестов (например, tearDown), а также методы для самого тестирования.

6. Какие методы класса TestCase выполняются при запуске и завершении работы тестов?

Методы класса TestCase, выполняющиеся при запуске и завершении работы тестов:

setUp(): вызывается перед каждым тестовым методом.

tearDown(): вызывается после каждого тестового метода.

setUpClass(): вызывается перед запуском первого тестового метода в классе.

tearDownClass(): вызывается после завершения всех тестов в классе.

7. Какие методы класса TestCase используются для проверки условий и генерации ошибок?

Методы класса TestCase для проверки условий и генерации ошибок включают assertEqual(), assertTrue(), assertFalse(), assertRaises() и многие другие.

8. Какие методы класса TestCase позволяют собирать информацию о самом тесте?

Методы класса TestCase для сбора информации о тесте не столь явно выражены, как методы для проверки, но можно использовать id(), shortDescription() для получения информации о тестовом случае.

9. Каково назначение класса TestSuite? Как осуществляется загрузка тестов?

Назначение класса TestSuite - группировка и последовательный запуск тестов. Загрузка тестов осуществляется через TestLoader или путем добавления тестовых случаев и наборов в TestSuite вручную.

10. Каково назначение класса TestResult?

Назначение класса TestResult - хранение и представление результатов тестов. Он используется в TestRunner для сбора информации о прохождении тестов, включая количество успешных, неудачных и пропущенных тестов.

11. Для чего может понадобиться пропуск отдельных тестов?

Пропуск отдельных тестов может понадобиться, если тест временно неприменим, требует еще разработки или зависит от условий, которые в данный момент не выполнены.

12. Как выполняется безусловный и условных пропуск тестов? Как выполнить пропуск класса тестов?

Безусловный пропуск: декоратор @unittest.skip("причина").

Условный пропуск: декораторы вроде @unittest.skipIf(condition, "причина").

Пропуск класса тестов: использование тех же декораторов на уровне класса.

13. Самостоятельно изучить средства по поддержке тестов unittest в PyCharm. Приведите обобщенный алгоритм проведения тестирования с помощью PyCharm.

Алгоритм проведения тестирования в PyCharm:

Создайте тестовый файл в своем проекте.

Используйте структуры unittest, например, классы TestCase, для написания тестов.

В PyCharm, щелкните правой кнопкой мыши на тестовом файле или тестовом методе и выберите "Run 'Unittests in <имя файла>'" для запуска тестов.

Посмотрите результаты во вкладке Run, где PyCharm покажет успешные тесты и тесты с ошибками, предоставив детальную информацию по каждому случаю.

Вывод: в результате выполнения работы были получены навыки по написанию автоматизированных тестов на языке программирования Python версии 3.x.