Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №1
дисциплины «Объектно-ориентированное программирование»

Выполнил:
Плугатырев Владислав Алексеевич
3 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка и сопровождение программного обеспечения», очная форма обучения

_____
(подпись)

Проверил Воронкин Роман Александрович
_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Тестированиев Python [unittest].

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

Ход работы.

1. Создание репозитория.

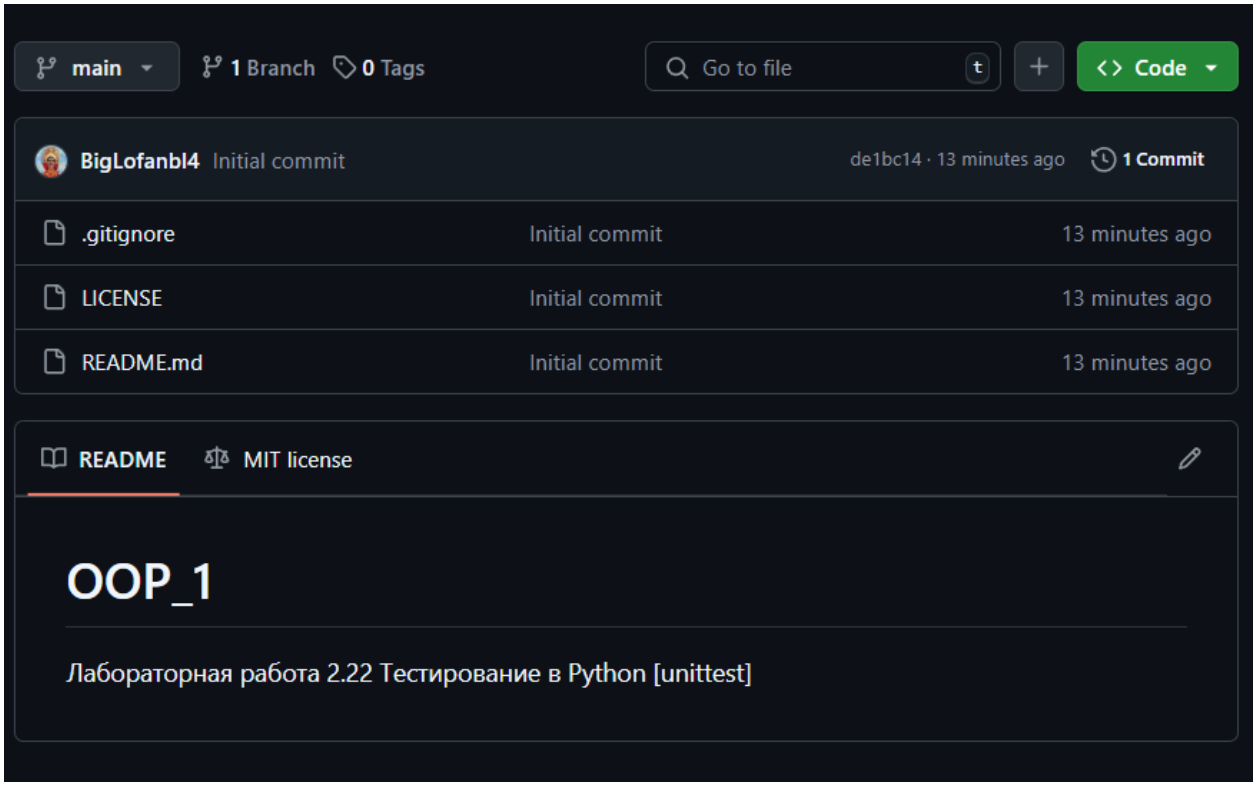

Рисунок 1.1 – Созданный репозиторий

2. Выполнение индивидуального задания: «Для индивидуального задания лабораторной работы 2.21 добавьте тесты с использованием модуля unittest, проверяющие операции по работе с базой данных.».

```python
class DatabaseOperationsTest(unittest.TestCase):
    # Перед началом каждого теста, создаем временную бд
    def setUp(self):
        self.db_path = Path("test_people.db")
        create_db(self.db_path)

    def tearDown(self):
        if self.db_path.exists():
            os.remove(self.db_path)

    # Проверяем правильно ли создаются таблицы
    def test_create_db(self):
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
        tables = cursor.fetchall()

        self.assertIn(("zodiacs",), tables)
        self.assertIn(("people",), tables)

        conn.close()

    def test_add_person(self):
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()

        surname = "Plugatyrev"
        name = "Vladislav"
        zodiac = "Capricorn"
        birthday = datetime.strptime("2005-01-12", "%Y-%m-%d").date()

        add_person(self.db_path, surname, name, zodiac, birthday)

        cursor.execute(
            """
        SELECT people.surname, people.name, zodiacs.zodiac_title, people.birthday
        FROM people
        INNER JOIN zodiacs ON zodiacs.zodiac_id = people.zodiac_id
        """
        )
        row = cursor.fetchall()[0]

        conn.close()

        self.assertEqual(row[0], surname)
        self.assertEqual(row[1], name)
        self.assertEqual(row[2], zodiac)
        self.assertEqual(row[3], birthday.strftime("%Y-%m-%d"))

    def test_select_by_surname(self):
        add_person(
            self.db_path,
            "Plugatyrev",
            "Vladislav",
            "Capricorn",
            datetime.strptime("2005-01-12", "%Y-%m-%d").date(),
        )
        add_person(
            self.db_path,
            "Chickodan",
            "Alexey",
            "Lion",
            datetime.strptime("2003-08-08", "%Y-%m-%d").date(),
        )
        add_person(
            self.db_path,
            "Ivanov",
            "Ivan",
            "Gemini",
            datetime.strptime("2000-11-30", "%Y-%m-%d").date(),
        )

        target = select_by_surname(self.db_path, "Chickodan")

        self.assertEqual(target[0]["surname"], "Chickodan")

    def test_select_all(self):
        add_person(
            self.db_path, "Plugatyrev", "Vladislav", "Capricorn", "2005-01-12"
        )
        add_person(self.db_path, "Chickodan", "Alexey", "Lion", "2003-08-08")
        add_person(self.db_path, "Ivanov", "Ivan", "Gemini", "2000-11-30")

        people = select_all(self.db_path)

        self.assertEqual(len(people), 3)
        self.assertEqual(people[0]["surname"], "Plugatyrev")
        self.assertEqual(people[1]["surname"], "Chickodan")
        self.assertEqual(people[2]["surname"], "Ivanov")
```

Рисунок 1.2 – Код программы

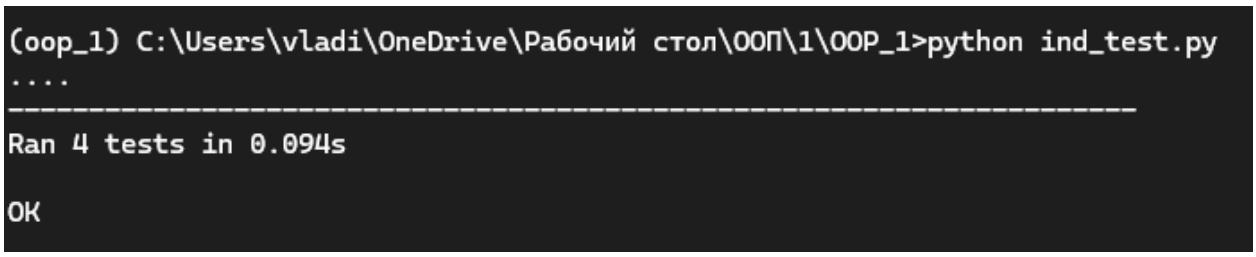

Рисунок 1.3 – Успешное выполнение тестов

Ответы на контрольные вопросы

1. Для чего используется автономное тестирование?

Автономное тестирование (или автоматизированное тестирование) используется для проверки программного обеспечения на наличие ошибок и соответствие требованиям без необходимости ручного тестирования. Оно позволяет быстро и эффективно проверять функциональность, производительность и другие аспекты программы, что снижает риск человеческих ошибок, ускоряет процесс тестирования и помогает поддерживать качество программного продукта.

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

В Python наибольшее распространение для автономного тестирования получили следующие фреймворки:

– «unittest»: Стандартный модуль для написания тестов в Python.

– «pytest»: Мощный фреймворк для тестирования, известный своей простотой и гибкостью.

– «nose2»: Развитие «nose», которое упрощает расширение возможностей «unittest».

– «doctest»: Позволяет тестировать документацию в строках документации (docstrings).

3. Какие существуют основные структурные единицы модуля unittest?

Основные структурные единицы модуля «unittest»:

– «TestCase»: Класс, в котором определяются тестовые методы.

– «TestSuite»: Класс для группировки тестов.

– «TestResult»: Класс для сбора результатов тестов.

– «TestLoader»: Класс для загрузки тестов.

– «TextTestRunner»: Класс для выполнения тестов и отображения результатов.

4. Какие существуют способы запуска тестов unittest?

Способы запуска тестов «unittest»:

– Запуск через командную строку: «python -m unittest <module>«, где «<module>« — это имя модуля с тестами.

– Запуск тестов из кода: используя методы «unittest.TextTestRunner» и «unittest.TestLoader».

– Использование встроенных функций IDE, таких как PyCharm, для запуска тестов.

5. Каково назначение класса TestCase?

Класс «TestCase» предназначен для создания тестов. Он предоставляет методы, которые можно переопределить для тестирования различных аспектов кода. Каждый метод класса «TestCase», имя которого начинается с «test», представляет собой отдельный тест.

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

– «setUp()»: Выполняется перед запуском каждого теста. Используется для подготовки окружения (например, создание временных файлов, установление соединений с базой данных).

– «tearDown()»: Выполняется после завершения каждого теста. Используется для очистки после теста (например, удаление временных файлов, закрытие соединений).

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

– «assertEqual(a, b)»: Проверяет, что «a» и «b» равны.
– «assertNotEqual(a, b)»: Проверяет, что «a» и «b» не равны.
– «assertTrue(x)»: Проверяет, что «x» истинно.
– «assertFalse(x)»: Проверяет, что «x» ложно.
– «assertIsNone(x)»: Проверяет, что «x» равен «None».
– «assertIsNotNone(x)»: Проверяет, что «x» не равен «None».
– «assertRaises(exception)»: Проверяет, что при выполнении блока кода возникает исключение «exception».

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

– «shortDescription()»: Возвращает краткое описание теста. Используется для генерации отчетов.
– «id()»: Возвращает уникальный идентификатор теста.
– «description»: Атрибут, который может быть установлен для задания описания теста.

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

Класс «TestSuite» используется для группировки нескольких тестов в один набор, который затем можно выполнить как единое целое. Это позволяет организовать тесты и выполнять их вместе.

Загрузка тестов осуществляется через класс «TestLoader», который предоставляет методы для загрузки тестов из модулей и классов.

10. Каково назначение класса TestResult?

Класс «TestResult» предназначен для сбора и хранения результатов выполнения тестов. Он собирает информацию о том, какие тесты прошли успешно, какие завершились с ошибками и какие были пропущены.

11. Для чего может понадобиться пропуск отдельных тестов?

Пропуск тестов может понадобиться, если тесты зависят от внешних ресурсов или условий, которые в данный момент недоступны. Также это может быть полезно, если тесты неактуальны или еще не готовы к выполнению.

12. Как выполняется безусловный и условные пропуски тестов? Как выполнить пропуск класса тестов?

Безусловный пропуск тестов: Используйте декоратор «@unittest.skip»:

Условный пропуск тестов: Используйте декораторы «@unittest.skipIf» или «@unittest.skipUnless».

Пропуск класса тестов: Используйте декоратор «@unittest.skip» на уровне класса.

13. Средства по поддержке тестов unittest в PyCharm. Обобщенный алгоритм проведения тестирования с помощью PyCharm:

В PyCharm поддержка тестов «unittest» осуществляется через встроенные инструменты тестирования. Вот обобщенный алгоритм:

1. Создание тестов: Напишите тесты, используя «unittest» или другой поддерживаемый фреймворк, и сохраните их в проекте.

2. Запуск тестов: Щелкните правой кнопкой мыши на файле с тестами или на папке с тестами в проекте, затем выберите «Run 'Unittests'« или «Run '<test_file>'».

3. Просмотр результатов: Результаты тестов будут отображаться в панели "Run" в нижней части PyCharm. Здесь можно увидеть, какие тесты прошли, какие завершились с ошибками, и какие были пропущены.

4. Исправление ошибок: При необходимости исправьте код и перезапустите тесты, чтобы убедиться, что исправления работают.