

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 №8**  
**дисциплины «анализ данных»**

Выполнил:  
Середа Кирилл Витальевич  
2 курс, группа ИВТ-б-о-22-1,  
09.03.01 «Информатика и  
вычислительная техника», очная  
форма обучения

---

(подпись)

Руководитель практики:  
Воронкин Роман Александрович

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2024 г.

Тема: Тестирование в Python [unittest]

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

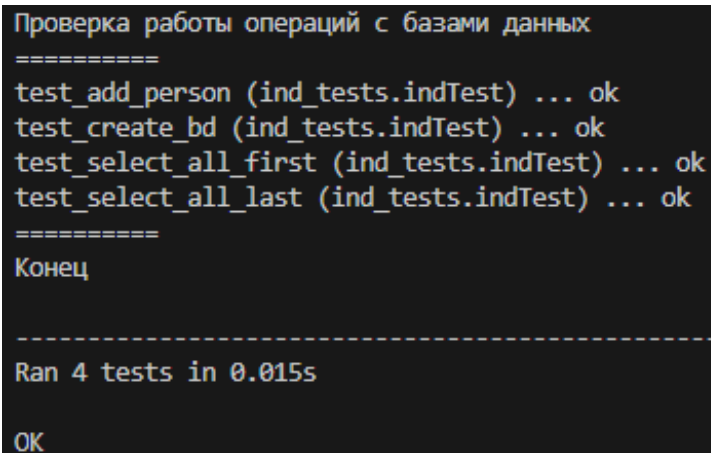
Ход работы:

Индивидуальное задание

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

Создал файл ind\_test.py, в котором написал различные тесты для проверки работы 2.21: проверка на создание бд, проверка на добавление элемента в бд, проверка на валидность количества столбцов, на рис. 2 представлен код.

Также создал файл ind\_runner.py, который запускает тест из файла ind\_test.py, на рис. 3 представлен код

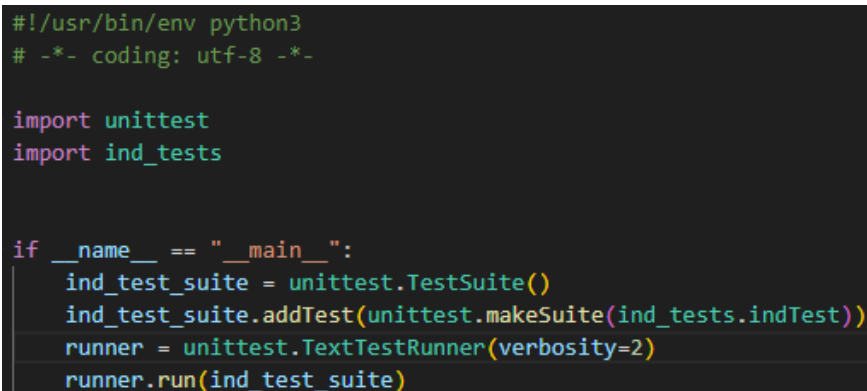


```
Проверка работы операций с базами данных
=====
test_add_person (ind_tests.indTest) ... ok
test_create_bd (ind_tests.indTest) ... ok
test_select_all_first (ind_tests.indTest) ... ok
test_select_all_last (ind_tests.indTest) ... ok
=====
Конец

-----
Ran 4 tests in 0.015s

OK
```

Рисунок 1 – Результат выполнения ind\_runner



```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import unittest
import ind_tests

if __name__ == "__main__":
    ind_test_suite = unittest.TestSuite()
    ind_test_suite.addTest(unittest.makeSuite(ind_tests.indTest))
    runner = unittest.TextTestRunner(verbosity=2)
    runner.run(ind_test_suite)
```

Рисунок 2 – Код ind\_runner.py

```

class indTest(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        """Set up for class"""
        print("Проверка работы операций с базами данных")
        print("=====")

    @classmethod
    def tearDownClass(cls):
        """Tear down for class"""
        print("=====")
        print("Конец")

    def test_create_bd(self):
        def created_bd(name_bd):
            file_path = pathlib.Path.cwd() / name_bd

            if file_path.exists and file_path.is_file:
                return True
            return False

        self.assertEqual(ind.create_db("test_bd"), created_bd("test_bd"))

    def test_add_person(self):
        def added_person(name_bd, full_name, birth_date, phone):
            conn = sqlite3.connect(name_bd)
            cursor = conn.cursor()

            cursor.execute(
                """
                SELECT names.full_name, person.birth_date, person.phone_number
                FROM person
                INNER JOIN names ON names.name_id = person.name_id
                WHERE
                names.full_name = ? and
                person.birth_date = ? and
                person.phone_number = ?
                """
            )
            rows = cursor.fetchall()
            conn.close()

            return bool(rows)

        ind.add_person("test_bd", "Артём", 22, 5555)
        self.assertEqual(
            True,
            added_person("test_bd", "Артём", 22, 5555),
        )

    def test_select_all_last(self):

```

Рисунок 3 – Код ind\_test.py

Вывод: в ходе выполнения работы были приобретены навыки написания автоматизированных тестов на языке программирования Python версии 3.x.

Контрольные вопросы:

## 1. Назначение автономного тестирования

Автономное тестирование используется для автоматизации процесса проверки работоспособности программного обеспечения без необходимости постоянного вмешательства человека. Оно позволяет проводить тестирование в автоматическом режиме, что ускоряет процесс разработки и обеспечивает более надежное тестирование приложений.

## 2. Фреймворки Python для автономного тестирования

Наиболее распространённые фреймворки Python для решения задач автономного тестирования включают unittest и pytest.

## 3. Основные структурные единицы модуля unittest

Основные структурные единицы модуля unittest в Python включают в себя классы TestCase, TestSuite и TestResult.

## 4. Способы запуска тестов unittest

Тесты модуля unittest могут быть запущены с помощью методов командной строки, IDE или специальных инструментов для автоматизации тестирования.

## 5. Назначение класса TestCase

Класс TestCase в модуле unittest предназначен для создания тестовых случаев, которые могут быть запущены в автономном режиме для проверки различных аспектов функциональности программного обеспечения.

## 6. Методы класса TestCase при запуске и завершении работы тестов

При запуске и завершении работы тестов, класс TestCase выполняет методы setUp() и tearDown() соответственно.

## 7. Методы класса TestCase для проверки условий и генерации ошибок

Для проверки условий и генерации ошибок, методы assertEquals(), assertTrue(), assertFalse() и другие аналогичные методы используются в классе TestCase.

## 8. Методы класса TestCase для сбора информации о тесте

Для сбора информации о самом тесте, в классе TestCase используются методы setUpClass() и tearDownClass().

## 9. Назначение класса TestSuite и его загрузка

Класс TestSuite в модуле unittest предназначен для объединения тестовых случаев. Загрузка тестов в TestSuite осуществляется путем добавления тестовых случаев с помощью метода addTest().

## 10. Назначение класса TestResult

Класс TestResult в модуле unittest предназначен для хранения результатов выполнения тестов.

## 11. Пропуск отдельных тестов

Пропуск отдельных тестов может понадобиться, например, при временной неработоспособности определенной функциональности или при необходимости отложить проверку определенного теста на будущее.

## 12. Безусловный и условный пропуск тестов

Безусловный пропуск тестов выполняется с помощью декоратора @unittest.skip(), а условный пропуск – с помощью декоратора @unittest.skipIf() или @unittest.skipUnless(). Для пропуска класса тестов используется декоратор @unittest.skip() перед объявлением класса.

13. Для проведения тестирования с помощью PyCharm можно использовать следующий обобщенный алгоритм:

1) Настройка проекта: Убедитесь, что ваш проект настроен в PyCharm и все необходимые зависимости установлены.

2) Создание тестовых случаев: Используйте модуль unittest для создания тестовых случаев в вашем проекте.

3) Запуск тестов: В PyCharm можно запустить тесты, выбрав соответствующий тестовый файл или директорию с тестами, и затем запустив их с помощью контекстного меню или специальных команд.

4) Анализ результатов: После запуска тестов в PyCharm, вы можете проанализировать результаты выполнения тестов и увидеть отчет о прохождении тестов.

5) Отладка тестов (при необходимости): При возникновении ошибок в тестах, вы можете использовать возможности отладки в PyCharm для их исправления.

6) Интеграция с другими инструментами (при необходимости): При необходимости, вы можете интегрировать PyCharm с другими инструментами для автоматизации тестирования, такими как nose или pytest.