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»
Вариант №2

Выполнила:
Беседина Инга Олеговна
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника», очная
форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р. А., канд. технических
наук, доцент, доцент кафедры
инфокоммуникаций

_____
(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.

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


"""
Вариант 2.
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 students import create_db, add_student, select_all, select_students


class CreateDbTest(unittest.TestCase):
    def setUp(self):
        self.test_path = Path("db_test.db")

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

    def test_create_db(self):
        create_db(self.test_path)
        self.assertTrue(self.test_path.exists())

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

        cursor.execute(
            """SELECT name FROM sqlite_master
            WHERE type='table' AND name='groups';"""
        )
```

```
        self.assertTrue(cursor.fetchone())

        cursor.execute("PRAGMA table_info(groups);")
        columns = cursor.fetchall()
        expected_columns = [
            (0, "group_id", "INTEGER", 0, None, 1),
            (1, "group_title", "TEXT", 1, None, 0),
        ]
        self.assertEqual(columns, expected_columns)

        cursor.execute(
            """SELECT name FROM sqlite_master
            WHERE type='table' AND name='students';"""
        )
        self.assertTrue(cursor.fetchone())

        cursor.execute("PRAGMA table_info(students);")
        columns = cursor.fetchall()
        expected_columns = [
            (0, "student_id", "INTEGER", 0, None, 1),
            (1, "student_name", "TEXT", 1, None, 0),
            (2, "group_id", "INTEGER", 1, None, 0),
            (3, "student_grades", "TEXT", 1, None, 0),
        ]
        self.assertEqual(columns, expected_columns)

        cursor.close()
        conn.close()


class TestStudents(unittest.TestCase):
    def setUp(self):
        self.test_db = Path("db_test.db")
        create_db(self.test_db)

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

    def test_add_student(self):
        add_student(self.test_db,
                    "Захарова В.Г", "ЮРП-б-о-23-2", "4,5,5,4,5")
        add_student(self.test_db,
                    "Волков О.Д", "ЭКМ-б-о-20-1", "5,5,5,4,5")

        conn = sqlite3.connect(self.test_db)
        cursor = conn.cursor()
```

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

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

        cursor.execute(
            """
            SELECT students.student_name, groups.group_title,
            students.student_grades FROM students
            INNER JOIN groups ON groups.group_id = students.group_id
            """
        )
        rows = cursor.fetchall()

        cursor.close()
        conn.close()

        self.assertEqual(len(rows), 2)
        self.assertDictEqual(
            {
                "name": rows[0][0],
                "group": rows[0][1],
                "grades": rows[0][2],
            },
            {
                "name": "Захарова В.Г",
                "group": "ЮРП-б-о-23-2",
                "grades": "4,5,5,4,5",
            },
        )
        self.assertDictEqual(
            {
                "name": rows[1][0],
                "group": rows[1][1],
                "grades": rows[1][2],
            },
            {
                "name": "Волков О.Д",
                "group": "ЭКМ-б-о-20-1",
                "grades": "5,5,5,4,5",
            },
        )

    def test_select_all(self):
        add_student(self.test_db,
                    "Захарова В.Г", "ЮРП-б-о-23-2", "4,5,5,4,5")
```

```
        add_student(self.test_db,
                    "Волков О.Д", "ЭКМ-б-о-20-1", "5,5,5,4,5")

        students = select_all(self.test_db)
        self.assertEqual(len(students), 2)
        self.assertDictEqual(
            students[0],
            {
                "name": "Захарова В.Г",
                "group": "ЮРП-б-о-23-2",
                "grades": "4,5,5,4,5",
            },
        )
        self.assertDictEqual(
            students[1],
            {
                "name": "Волков О.Д",
                "group": "ЭКМ-б-о-20-1",
                "grades": "5,5,5,4,5",
            },
        )

    def test_select_students(self):
        add_student(self.test_db,
                    "Васильева М.А", "ЭКМ-б-о-21-1", "5,5,3,4,5")
        add_student(self.test_db,
                    "Захарова В.Г", "ЮРП-б-о-23-2", "4,5,5,4,5")
        add_student(self.test_db,
                    "Волков О.Д", "ЭКМ-б-о-20-1", "5,5,5,4,5")

        students = select_students(self.test_db)
        self.assertEqual(len(students), 2)
        self.assertDictEqual(
            students[0],
            {
                "name": "Захарова В.Г",
                "group": "ЮРП-б-о-23-2",
                "grades": "4,5,5,4,5",
            },
        )
        self.assertDictEqual(
            students[1],
            {
                "name": "Волков О.Д",
                "group": "ЭКМ-б-о-20-1",
                "grades": "5,5,5,4,5",
            },
```

```
        )


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

Модуль test_runner.py:

```
import unittest
import utest_students


testLoad = unittest.TestLoader()
suites = testLoad.loadTestsFromModule(utest_students)
runner = unittest.TextTestRunner(verbosity=2)
runner.run(suites)
```

```
(venv) PS C:\Rep\DA_8\Project> python test_runner.py
test_create_db (utest_students.CreateDbTest.test_create_db) ... ok
test_add_student (utest_students.TestStudents.test_add_student) ... ok
test_select_all (utest_students.TestStudents.test_select_all) ... ok
test_select_students (utest_students.TestStudents.test_select_students) ... ok

----------------------------------------------------------------------
Ran 4 tests in 1.689s

OK
```

Рисунок 1. Результат запуска тестов

Контрольные вопросы:

1. Автономное тестирование используется для проверки правильности работы программного кода. Оно позволяет автоматизировать процесс проверки функций, классов или модулей на соответствие ожидаемому поведению. Автономные тесты помогают выявить ошибки и обеспечивают уверенность в работоспособности кода при его изменении или рефакторинге.

2. В Python наиболее популярными фреймворками для автономного тестирования являются:

- unittest (встроенный модуль в стандартную библиотеку Python)
- pytest
- nose

- doctest

3. Основными структурными единицами модуля unittest являются:

- TestCase: класс, в котором определяются тестовые методы.

- TestSuite: класс, представляющий собой коллекцию тестовых методов или других тестовых наборов.

- TestLoader: класс, отвечающий за загрузку тестов из модулей и сбор их в TestSuite.

- TestResult: класс, который содержит результаты выполнения тестов.

4. Тесты модуля unittest можно запускать несколькими способами:

- Запуск через командную строку с использованием утилиты unittest или pytest.

- Запуск тестов непосредственно из кода с использованием функции unittest.main().

- Запуск тестов с помощью интегрированных сред разработки, таких как PyCharm или Visual Studio Code.

5. Класс TestCase в модуле unittest используется для создания тестовых случаев. Он предоставляет базовый функционал для написания и организации тестов.

6. При запуске и завершении работы тестовых методов класса TestCase выполняются следующие методы:

- setUpClass() и tearDownClass(): выполняются перед началом и после окончания всех тестовых методов в классе.

- setUp() и tearDown(): выполняются перед началом и после каждого тестового метода в классе.

7. Для проверки условий и генерации ошибок в классе TestCase используются следующие методы:

- assertEqual(a, b): проверяет, что значения a и b равны.

- assertTrue(x): проверяет, что значение x является истинным.

- assertFalse(x): проверяет, что значение x является ложным.

- assertRaises(exception, callable, *args, kwargs): проверяет, что вызов callable(*args, kwargs) генерирует исключение exception.

8. Методы класса TestCase, которые позволяют собирать информацию о самом тесте, включают:

- setUp() и tearDown(): можно использовать для подготовки данных перед выполнением теста и очистки после его выполнения.

- setUpClass() и tearDownClass(): можно использовать для подготовки данных перед выполнением всех тестов в классе и очистки после выполнения всех тестов.

9. Класс TestSuite используется для группировки тестовых методов или других тестовых наборов. Загрузка тестов происходит с помощью класса TestLoader, который находит и загружает тесты из модулей и собирает их в TestSuite. Тесты могут быть добавлены в TestSuite как отдельные методы или целые классы.

10. Класс TestResult представляет собой контейнер для хранения результатов выполнения тестов. Он отслеживает количество успешных и неудачных тестов, а также содержит информацию об ошибках и отчеты о выполнении тестов.

11. Пропуск отдельных тестов может понадобиться, например, когда некоторые условия не выполняются или когда определенные зависимости недоступны. Пропуск теста означает, что он не будет выполнен, но не будет считаться ошибкой.

12. В модуле unittest можно выполнять безусловный пропуск теста с помощью декоратора unittest.skip(reason). Условный пропуск теста можно выполнить с помощью декоратора unittest.skipIf(condition, reason), где condition — это условие, при котором происходит пропуск. Чтобы пропустить весь класс тестов, можно использовать декоратор unittest.skip(reason) перед объявлением класса.

13. PyCharm предоставляет удобные инструменты для поддержки тестирования с помощью модуля unittest. Общий алгоритм проведения тестирования в PyCharm может быть следующим:

1. Создайте новый файл с расширением .py для написания тестов.

2. Импортируйте модуль unittest.

3. Определите класс на основе unittest. TestCase и определите в нем методы для проведения тестирования.

4. Запустите тесты одним из следующих способов:

- Щелкните правой кнопкой мыши на файле с тестами в структуре проекта и выберите "Run Unittests in <filename>".

- Используйте комбинацию клавиш Ctrl+Shift+F10 (или Shift+F10 для повторного запуска последнего запуска).

- Используйте окно "Run" или "Debug" для запуска или отладки конкретного теста или всего файла с тестами.

- Используйте консольный режим запуска командой python -m unittest <filename>.py.

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