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

Институт цифрового развития
Кафедра инфокоммуникаций

«Лабораторная работа 2.3. Тестирование

в Python [unittest]»

ОТЧЕТ
по лабораторной работе №25
дисциплины
«Основы программной инженерии»

Выполнил:
Луценко Дмитрий Андреевич
2 курс, группа ПИЖ-б-о-21-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

_____
(подпись)

Проверил:
_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Лабораторная работа 23. Тестирование в Python [unittest]

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

Ход работы:

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

import pathlib
from pathlib import Path
import sqlite3
import unittest
from unittest.mock import patch
from ind import create_db, add_product, select_all, select_by_shop


class ProdTest(unittest.TestCase):
    """
    Тест программы для списка продуктов
    """

    @classmethod
    def setUpClass(cls):
        """Set up for class"""
        print("setUpClass")
        print("==========")

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

    def setUp(self):
        """Set up for test"""
        print("Set up for [" + self.shortDescription() + "]")


    def tearDown(self):
        """Tear down for test"""
        print("Tear down for [" + self.shortDescription() + "]")
        print("")


    def test_create_db(self):
        """
        Проверка создания БД.
        """
        database_path = "test.db"
        if Path(database_path).exists():
            Path(database_path).unlink()

        create_db(database_path)
        self.assertTrue(Path(database_path).is_file())
        Path(database_path).unlink()


    def test_add_product(self):
        """
        Проверка добавления записи о товаре.
```

```
        """
        database_path = "test.db"
        create_db(database_path)
        add_product(database_path, 'молоко', 'пятерочка', 55.9)
        conn = sqlite3.connect(database_path)
        cursor = conn.cursor()
        cursor.execute(
            """
            SELECT * FROM prods
            """
        )
        row = cursor.fetchone()
        self.assertEqual(row, (1, 'молоко', 1, 55.9))
        conn.close()
        Path(database_path).unlink()


    def test_select_all(self):
        """
        Проверка выбора всего списка
        """
        database_path = "test.db"
        create_db(database_path)
        add_product(database_path, 'хлеб', 'магнит', 30.0)
        add_product(database_path, 'ноутбук', 'ситилинк', 50000.0)
        add_product(database_path, 'макароны', 'магнит', 44.4)

        r_output = [
            {'prod': 'хлеб', 'shop': 'магнит', 'cost': 30.0},
            {'prod': 'ноутбук', 'shop': 'ситилинк', 'cost': 50000.0},
            {'prod': 'макароны', 'shop': 'магнит', 'cost': 44.4}
        ]
        self.assertEqual(select_all(database_path), r_output)
        Path(database_path).unlink()


    def test_select_by_shop(self):
        """
        Проверка выбора товаров из одного магазина
        """
        database_path = "test.db"
        create_db(database_path)
        add_product(database_path, 'хлеб', 'магнит', 30.0)
        add_product(database_path, 'ноутбук', 'ситилинк', 50000.0)
        add_product(database_path, 'макароны', 'магнит', 44.4)
        r_output = [
            {'prod': 'хлеб', 'shop': 'магнит', 'cost': 30.0},
            {'prod': 'макароны', 'shop': 'магнит', 'cost': 44.4}
        ]
        self.assertEqual(select_by_shop(database_path, 'магнит'), r_output)
        Path(database_path).unlink()
```

Листинг 1 – unittest’ы

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

```
import unittest
import ind_tests


prodTestSuite = unittest.TestSuite()
prodTestSuite.addTest(unittest.makeSuite(ind_tests.ProdTest))
runner = unittest.TextTestRunner(verbosity=2)
runner.run(prodTestSuite)
```

Листинг 2 – test_runner

Ответы на контрольные вопросы:

1. Для чего используется автономное тестирование? Автономный тест – это автоматизированная часть кода, которая вызывает тестируемую единицу работы и затем проверяет некоторые предположения о единственном конечном результате этой единицы. Автономное тестирование применяется для тестирования функций, классов, методов и т.д. с целю выявления ошибок в работе в этих отдельных единицах общей программы.

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

- unittest;
- nose;
- pytest.

3. Какие существуют основные структурные единицы модуля unittest?

Test fixture

Test fixture – обеспечивает подготовку окружения для выполнения тестов, а также организацию мероприятий по их корректному завершению (например очистка ресурсов). Подготовка окружения может включать в себя создание баз данных, запуск необходим серверов и т.п.

Test case

Test case – это элементарная единица тестирования, в рамках которой проверяется работа компонента тестируемой программы (метод, класс, поведение и т. п.). Для реализации этой сущности используется класс *TestCase*.

Test suite

Test suite – это коллекция тестов, которая может в себя включать как отдельные *test case'ы* так и целые коллекции (т.е. можно создавать коллекции коллекций). Коллекции используются с целью объединения тестов для совместного запуска.

Test runner

Test runner – это компонент, которые оркестрирует (координирует взаимодействие) запуск тестов и предоставляет пользователю результат их выполнения. *Test runner* может иметь графический интерфейс, текстовый интерфейс или возвращать какое-то заранее заданное значение, которое будет описывать результат прохождения тестов.

4. Какие существуют способы запуска тестов unittest? Запуск тестов можно сделать как из командной строки, так и с помощью графического интерфейса пользователя (GUI).

5. Каково назначение класса TestCase? Он представляет собой класс, который должен являться базовым для всех остальных классов, методы которых будут тестировать те или иные автономные единицы исходной программы. Для того, чтобы метод класса выполнялся как тест, необходимо, чтобы он начинался со слова test.

6. Какие методы класса TestCase выполняются при запуске и завершении работы тестов? setUp() Метод вызывается перед запуском теста. Как правило, используется для подготовки окружения для теста.

tearDown() Метод вызывается после завершения работы теста. Используется для “приборки” за тестом.

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

Метод	Описание
`assertEqual(a, b)`	a == b
`assertNotEqual(a, b)`	a != b
`assertTrue(x)`	bool(x) is True
`assertFalse(x)`	bool(x) is False
`assertIs(a, b)`	a is b
`assertIsNot(a, b)`	a is not b
`assertIsNone(x)`	x is None
`assertIsNotNone(x)`	x is not None
`assertIn(a, b)`	a in b
`assertNotIn(a, b)`	a not in b
`assertIsInstance(a, b)`	isinstance(a, b)
`assertNotIsInstance(a, b)`	not isinstance(a, b)

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

countTestCases() Возвращает количество тестов в объекте класса-наследника от TestCase.

id() Возвращает строковый идентификатор теста. Как правило это полное имя метода, включающее имя модуля и имя класса.

shortDescription() Возвращает описание теста, которое представляет собой первую строку docstring’а метода, если его нет, то возвращает None.

9. Каково назначение класса TestSuite? Как осуществляется загрузка тестов? Класс TestSuite используется для объединения тестов в группы, которые могут включать в себя как отдельные тесты так и заранее созданные группы. Помимо этого, TestSuite предоставляет интерфейс, позволяющий TestRunner’у, запускать тесты.

10. Каково назначение класса TestResult? Класс TestResult используется для сбора информации о результатах прохождения тестов.

11. Для чего может понадобиться пропуск отдельных тестов? Во избежание ошибок тестирования, так как некоторые тесты могут давать заведомо неправильный результат в зависимости от какого-либо условия. Для этого такие тесты необходимо пропускать.

12. Как выполняется безусловный и условных пропуск тестов? Как выполнить пропуск класса тестов? Безусловный пропуск: @unittest.skip(reason) записывается перед объявлением теста. Условный пропуск: 1) @unittest.skipIf(condition, reason) – Тест будет пропущен, если условие (condition) истинно. 2) @unittest.skipUnless(condition, reason) – Тест будет пропущен если, условие (condition) не истинно. Пропуск класса тестов: @unittest.skip(reason) записывается перед объявлением класса.

13. Самостоятельно изучить средства по поддержке тестов unittest в PyCharm. Приведите обобщенный алгоритм проведения тестирования с помощью PyCharm. В PyCharm есть встроенная поддержка unit тестов, которая позволяет создавать шаблон класса для тестирования и его дальнейшей настройки. 1) Необходимо создать класс для тестирования. 2) Написание кода тестов в классе для тестирования частей программы. 3) Запуск тестов 4) Debug тестов при необходимости. 5) Автоматизация тестов. PyCharm поддерживает автоматизацию тестов – установив её, вы можете сфокусироваться в написании кода самой программы, а IDE будет в автоматическом режиме проводить тестирование по мере изменения кода.