Институт цифрового развития
Кафедра инфокоммуникаций

«Тестирование в Python [unittest]»

ОТЧЕТ
по лабораторной работе №22
дисциплины
«Основы программной инженерии»

Выполнил:
Мизин Глеб Егорович
2 курс, группа ПИЖ-б-о-21-1,
011.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

_____
(подпись)

Проверил:
_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

```python
import sqlite3
import unittest
from data_base_prog import Path, create_db, add_account, sum_check


class TestCreateDB(unittest.TestCase):

    def setUp(self):
        self.database_path = Path('test.db')

    def test_create_db(self):
        create_db(self.database_path)
        conn = sqlite3.connect(self.database_path)
        cursor = conn.cursor()
        cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name='Transfers'")
        result = cursor.fetchone()
        self.assertEqual(result, ('Transfers',))
        cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name='Accounts'")
```

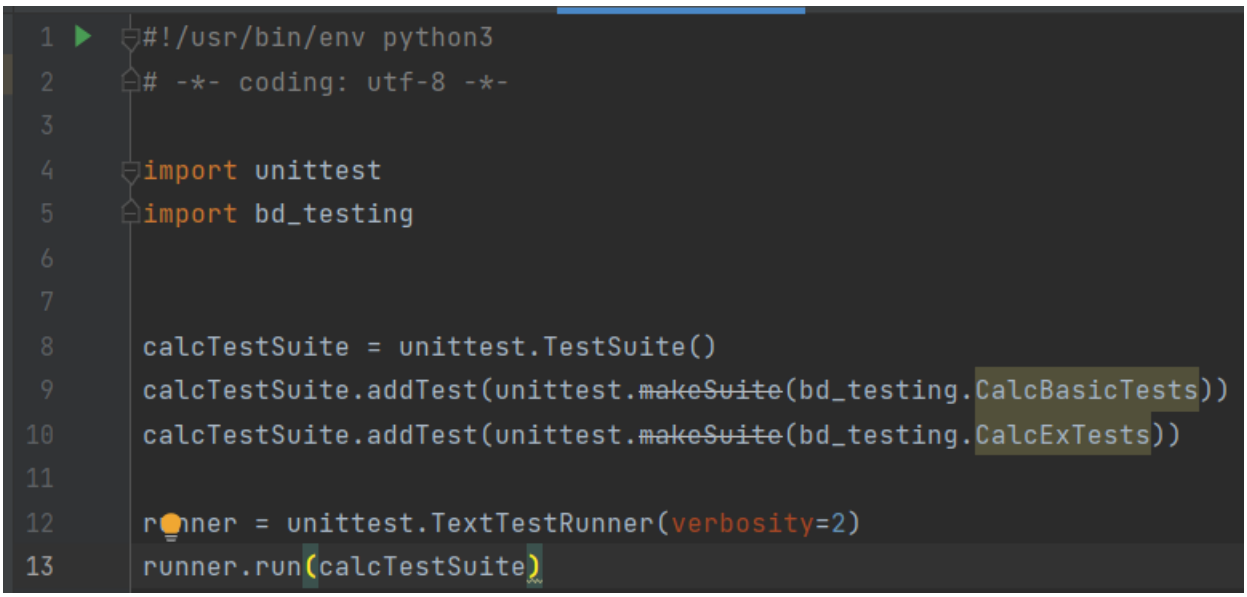
```python
def test_add_account(self):
    add_account(self.database_path, Sender=1, Receiver=2, Transfer_ammount=100)
    conn = sqlite3.connect(self.database_path)
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM Accounts")
    result = cursor.fetchall()
    conn.close()

    # Проверяем ID, отправителя и получателя в первой бд
    self.assertEqual(result[0][1], 1)
    self.assertEqual(result[0][2], 1)
    self.assertEqual(result[0][3], 2)

    conn = sqlite3.connect(self.database_path)
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM Transfers")
    result = cursor.fetchall()
    conn.close()

    # Проверяем сумму трансфера
    self.assertEqual(result[0][1], 100)

def tearDown(self):
    self.database_path.unlink()
```



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

import unittest
import bd_testing


calcTestSuite = unittest.TestSuite()
calcTestSuite.addTest(unittest.makeSuite(bd_testing.CalcBasicTests))
calcTestSuite.addTest(unittest.makeSuite(bd_testing.CalcExTests))

runner = unittest.TextTestRunner(verbosity=2)
runner.run(calcTestSuite)
```

Контрольные вопросы

1. Для чего используется автономное тестирование?

Автономное тестирование используется для автоматического тестирования программного обеспечения без участия человека, что упрощает нахождение и исправление ошибок.

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

Наиболее распространенные фреймворки для автономного тестирования на Python - unittest, pytest и nose.

3. Какие существуют основные структурные единицы модуля unittest?

Основными структурными единицами модуля unittest являются классы TestCase, TestSuite, и TestResult.

4. Какие существуют способы запуска тестов unittest?

Тесты unittest можно запускать как из командной строки, так и из IDE, например, PyCharm.

5. Каково назначение класса TestCase?

Класс TestCase используется для определения тестовых кейсов, которые содержат набор проверок для конкретной функциональности.

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

Методы setUp и tearDown выполняются перед и после каждого тестового метода соответственно.

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

Методы assertEqual и assertRaises используются для проверки условий и генерации ошибок.

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

Методы addSuccess, addError, и addFailure позволяют собирать информацию о каждом тесте.

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

Класс TestSuite используется для группировки и запуска нескольких тестовых кейсов. Тесты загружаются при помощи метода addTest.

10. Каково назначение класса TestResult?

Класс TestResult используется для хранения результатов тестирования, включая количество пройденных и проваленных тестов, а также информацию об ошибках.

11. Для чего может понадобиться пропуск отдельных тестов?

Пропуск тестов может понадобиться, например, если функционал, для которого предназначен тест, находится в разработке или не может быть проверен автоматически.

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

Безусловный пропуск тестов происходит при помощи метода skipTest, а условный - при помощи метода skipIf или skipUnless. Чтобы пропустить класс тестов, можно использовать декоратор skip.

13. Приведите обобщенный алгоритм проведения тестирования с помощью PyCharm.

Для проведения тестов в PyCharm нужно создать новый проект, добавить в него файлы с тестами и запустить их с помощью кнопки "Run". Результаты тестирования будут отображаться в консоли и в специальной вкладке "Test Runner". В PyCharm также есть возможность использования отладчика для исправления ошибок.