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 №2.22
дисциплины «Анализ данных»

Выполнил:
Говоров Егор Юрьевич
2 курс, группа ИВТ-б-о-22-1,
3.05.2024 «Информатика и вычислительная техника»,
направленность (профиль) «Информатика и вычислительная техника», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р. А., доцент кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой_____ Дата защиты_____

Ставрополь, 2023 г.

Тема: тестирование в Python [unittest]

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

Ход работы:

Задание 1. Создал проект PyCharm в папке репозитория. Приступил к работе с примером. Добавил новый файл example.py.

Условие примера: Написать код, который будет проверять правильность работы файла calc_test.py

```
D:\Users\Admin\Рабочий стол\Data_Analize\Data_Analize_Govorov-8\example>python calc_test.py
....
----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
```

Рисунок 1. Выполнение первого примера

Индивидуальное задание

Вариант 5

Создал новый файл под названием test.py.

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

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

import argparse
import sqlite3
import typing as t
from pathlib import Path

DATABASE_FILE = "flights.db"

1 usage  new *
def add_flight(database_path: Path, destination: str, flight_number: str, aircraft_type: str) -> None:
    """
    Добавить новый рейс в базу данных.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()

    cursor.execute(
        __sql: """
        INSERT INTO flights (destination, flight_number, aircraft_type)
        VALUES (?, ?, ?)
        """,
        __parameters: (destination, flight_number, aircraft_type),
    )
    conn.commit()
    conn.close()

2 usages  new *
def print_flights(database_path: Path) -> None:
    """
```

Рисунок 4. Основной код

```python
import unittest
import sqlite3
from pathlib import Path
import os
import flight_management  # Assuming this is where your main code resides

TEST_DB = "test_flights.db"

1 usage  new *
class CustomTestResult(unittest.TextTestResult):
    new *
    def __init__(self, stream, descriptions, verbosity):
        super().__init__(stream, descriptions, verbosity)

    new *
    def addSuccess(self, test):
        super().addSuccess(test)
        self.stream.writeln(f"{self.getDescription(test)} ... ok")

    new *
    def addSkip(self, test, reason):
        super().addSkip(test, reason)
        self.stream.writeln(f"{self.getDescription(test)} ... skipped '{reason}'")


1 usage  new *
class CustomTestRunner(unittest.TextTestRunner):
    new *
    def _makeResult(self):
        return CustomTestResult(self.stream, self.descriptions, self.verbosity)
```

Рисунок 5. Код для индивидуального задания

```
D:\Users\Admin\Рабочий стол\Data_Analize\Data_Analize_Govorov-8\individ>python test.py -v
test_add_flight (__main__.FlightManagementTest.test_add_flight)
Test adding a flight. ... ok
test_add_flight (__main__.FlightManagementTest.test_add_flight)
Test adding a flight. ... ok
test_print_flights (__main__.FlightManagementTest.test_print_flights)
Test printing all flights. ... ok
test_print_flights (__main__.FlightManagementTest.test_print_flights)
Test printing all flights. ... ok
test_search_flights_by_aircraft_type (__main__.FlightManagementTest.test_search_flights_by_aircraft_type)
Test searching flights by aircraft type. ... ok
test_search_flights_by_aircraft_type (__main__.FlightManagementTest.test_search_flights_by_aircraft_type)
Test searching flights by aircraft type. ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.856s

OK
```

Рисунок 6. Результат выполнения

Ответы на контрольные вопросы:

1. Для чего используется автономное тестирование?

Автономное тестирование используется для автоматизации проверки функциональности программного обеспечения. Это позволяет эффективно и систематически проверять, что изменения в коде не приводят к нарушению существующих функций, а также обнаруживать ошибки на ранних стадиях разработки.

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

Наиболее популярные фреймворки для автономного тестирования на языке Python включают unittest, pytest, и nose. unittest является встроенным модулем, тогда как pytest и nose предоставляют дополнительные возможности и синтаксис для более удобного написания тестов.

3. Какие существуют основные структурные единицы модуля unittest?

Основными структурными единицами модуля unittest являются:

- TestCase: Класс, описывающий отдельный тестовый случай.
- TestSuite: Класс, который группирует тестовые случаи для их выполнения вместе.
- TestLoader: Класс, который автоматически находит и загружает тестовые случаи.
- TestResult: Класс, который собирает результаты выполнения тестов.

4. Какие существуют способы запуска тестов unittest?

Тесты unittest можно запускать из командной строки с использованием unittest модуля или внутри среды разработки, такой как PyCharm. Можно также использовать Test Discovery для автоматического обнаружения и запуска тестов.

5. Каково назначение класса TestCase?

Класс TestCase предназначен для создания отдельных тестовых случаев. Он предоставляет методы для установки и проверки предварительных

условий, а также для группировки тестов.

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

Методы setUp выполняются перед запуском каждого теста, а методы tearDown выполняются после завершения каждого теста.

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

Некоторые методы, используемые для проверки условий и генерации ошибок, включают assertEqual, assertTrue, assertFalse, assertRaises и другие.

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

Методы, такие как setUp и tearDown, могут использоваться для подготовки данных и ресурсов перед выполнением тестов, а также после их выполнения.

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

Класс TestSuite предназначен для группировки тестовых случаев. Загрузка тестов осуществляется с использованием TestLoader, который автоматически находит и загружает тесты на основе заданных критериев.

10. Каково назначение класса TestResult?

Класс TestResult предназначен для сбора и представления результатов выполнения тестов. Он хранит информацию о том, сколько тестов было выполнено успешно, сколько неудачно, а также может включать другие подробности, такие как время выполнения и стеки вызовов.

11. Для чего может понадобиться пропуск отдельных тестов?

Пропуск тестов может быть полезен, если выполнение теста невозможно из-за временных условий, зависимостей или других

обстоятельств. Пропуск позволяет временно исключить тест из выполнения без его удаления из набора тестов.

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

Безусловный пропуск теста выполняется с использованием декоратора unittest.skip("Причина пропуска"). Условный пропуск может быть выполнен с использованием unittest.skipIf или unittest.skipUnless с указанием условий. Пропуск целого класса тестов выполняется с использованием декоратора unittest.skip("Причина пропуска") перед определением класса тестов.

13. Самостоятельно изучить средства по поддержке тестов unittest в PyCharm. Приведите обобщенный алгоритм проведения тестирования с помощью PyCharm.

PyCharm предоставляет удобные средства для тестирования с использованием unittest. Обобщенный алгоритм проведения тестирования в PyCharm включает следующие шаги:

- Шаг 1: Создание тестового проекта

1. Открыть PyCharm и создать новый проект или открыть существующий.
2. Создать директорию для тестов.

- Шаг 2: Написание тестов

1. Создать файл с тестами (обычно файл с префиксом test_).
2. Определить классы тестов, унаследованные от unittest.TestCase.
3. Написать методы тестов внутри классов, используя методы assert для проверки условий.

- Шаг 3: Запуск тестов

1. Открыть файл с тестами.
2. Нажать правой кнопкой мыши и выбрать "Run 'pytest in test_file.py'"
3. Посмотреть результаты выполнения тестов в окне вывода.

- Шаг 4: Анализ результатов

1. После выполнения тестов, PyCharm предоставит подробные результаты в специальной вкладке "Run" внизу экрана.
2. Анализировать результаты успешных и неуспешных тестов, и, при необходимости, вносить исправления в код.

Вывод: приобрел навыки написания автоматизированных тестов на языке программирования Python версии 3.x.