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:

Пустяков Андрей Сергеевич

2 курс, группа ИВТ-б-о-22-1,

09.03.01 «Информатика и вычислительная техника (профиль) «Программное обеспечение средств вычислительной
техники и автоматизированных систем», очная форма обучения

_____

(подпись)

Руководитель практики:

Воронкин Р. А. кандидат технических наук, доцент, доцент кафедры инфокоммуникаций

_____

(подпись)

Ставрополь, 2024 г.

Тема: Тестирование в Python [unittest].

Цель: приобрести навыки написания автоматизированных тестов на языке программирования Python версии 3.x.

Ход работы:

Создание общедоступного репозитория на «GitHub», клонирование репозитория, редактирование файла «.gitignore», организация репозитория согласно модели ветвления «git flow» (рис. 1).

```
C:\Users\Andrey\Desktop\Анализ_данных\8_лаба\Analysis_data_laboratory_work_8>git flow init

Which branch should be used for bringing forth production releases?
   - main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
```

Рисунок 1 – Создание репозитория

Проработка примеров лабораторной работы:

Пример 1.

Необходимо создать модуль «calc.py». Код программы данного модуля:

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


def add(a, b):
    return a + b


def sub(a, b):
    return a-b


def mul(a, b):
    return a * b


def div(a, b):
    return a / b


def sqrt(a):
    return a**0.5
```

```
def pow(a, b):
    return a**b
```

Проработка примера с «CalcTest». Код программы данного примера:

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

import unittest
import calc


class CalcTest(unittest.TestCase):
    """
    Calc tests
    """

    @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_add(self):
        """
        Add operation test
        """
        print("id: " + self.id())
        self.assertEqual(calc.add(1, 2), 3)

    def test_sub(self):
        """
        Sub operation test
        """
        print("id: " + self.id())
        self.assertEqual(calc.sub(4, 2), 2)
```

```python
    def test_mul(self):
        """
        Mul operation test
        """
        print("id: " + self.id())
        self.assertEqual(calc.mul(2, 5), 10)

    def test_div(self):
        """
        Div operation test
        """
        print("id: " + self.id())
        self.assertEqual(calc.div(8, 4), 2)


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

Результаты работы программы примера 1 (рис. 2).

```
Testing started at 2:33 ...
Launching unittests with arguments python -m unittest C:\Users\Andrey\Desktop\Анализ

setUpClass
==========
==========
tearDownClass
Set up for [Add operation test]
id: example_1.CalcTest.test_add
Tear down for [Add operation test]

Set up for [Div operation test]
id: example_1.CalcTest.test_div
Tear down for [Div operation test]

Set up for [Mul operation test]
id: example_1.CalcTest.test_mul


Tear down for [Mul operation test]
Ran 4 tests in 0.064s

OK
```

Рисунок 2 – Результаты работы программы примера 1

Выполнение индивидуальных заданий:

Задание 1.

Необходимо для индивидуального задания 2.21 добавить тесты с использованием модуля «unittest», проверяющие операции по работе с базой данных.

Для своего варианта лабораторной работы 2.17 необходимо реализовать хранение данных в базе данных SQLite3. Информация в базе данных должна храниться не менее чем в двух таблицах.

Необходимо использовать словарь, содержащий следующие ключи: название пункта назначения; номер поезда; время отправления. Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в список, состоящий из словарей заданной структуры; записи должны быть упорядочены по времени отправления поезда; вывод на экран информации о поездах, направляющихся в пункт, название которого введено с клавиатуры; если таких поездов нет, выдать на дисплей соответствующее сообщение (Вариант 26 (7), работа 2.8).

Код модуля с тестами индивидуального задания:

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

import unittest
import sqlite3
from pathlib import Path
import os
import induvidual_1


TEST_DB = "test_trains.db"


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

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

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


class CustomTestRunner(unittest.TextTestRunner):
    def _makeResult(self):
```

```
        return CustomTestResult(self.stream, self.descriptions,
self.verbosity)


class TrainManagementTest(unittest.TestCase):

    def setUp(self):
        """Создание тестовой базы данных перед каждым тестированием."""
        self.database_path = Path(TEST_DB)
        induvidual_1.create_db(self.database_path)

    def tearDown(self):
        """Удаление тестовой базы данных после каждого теста."""
        os.remove(self.database_path)

    def test_add_train(self):
        """Тест добавления поездов"""
        # Добавление поезда, используя эту функцию
        departure_point = "Москва"
        number_train = "123A"
        time_departure = "12:00"
        destination = "Санкт-Петербург"
        induvidual_1.add_train(self.database_path, departure_point,
number_train, time_departure, destination)

        # Проверка добавления поезда
        conn = sqlite3.connect(str(self.database_path))
        cursor = conn.cursor()
        cursor.execute(
            """
            SELECT t1.station_name AS departure_point, trains.train_number,
trains.time_departure, t2.station_name AS destination
            FROM trains
            JOIN stations t1 ON t1.station_id = trains.departure_id
            JOIN stations t2 ON t2.station_id = trains.destination_id
            WHERE trains.train_number = ?
            """,
            (number_train,)
        )
        row = cursor.fetchone()
        conn.close()
        self.assertIsNotNone(row)
        self.assertEqual(row, (departure_point, number_train, time_departure,
destination))

    def test_select_all(self):
        """Проверка выбора всех поездов."""
        # Add trains using the function
        trains_data = [
            ("Москва", "123", "12:00", "Санкт-Петербург"),
            ("Казань", "456", "14:00", "Москва")
        ]
        for train_data in trains_data:
            induvidual_1.add_train(self.database_path, *train_data)

        # Проверка выбора
        trains = induvidual_1.select_all(self.database_path)
        self.assertEqual(len(trains), len(trains_data))
        for train_data in trains_data:
            self.assertIn({
                "number_train": train_data[1],
                "departure_point": train_data[0],
                "time_departure": train_data[2],
                "destination": train_data[3],
```

```
            }, trains)

    def test_select_by_destination(self):
        """Тестирование поездов по пункту назначения"""
        trains_data = [
            ("Москва", "123", "12:00", "Санкт-Петербург"),
            ("Казань", "456", "14:00", "Москва"),
            ("Сочи", "789", "16:00", "Москва")
        ]
        for train_data in trains_data:
            induvidual_1.add_train(self.database_path, *train_data)

        # Проверка поезда на пункт назначения Москава.
        selected_trains =
induvidual_1.select_by_destination(self.database_path, "Москва")
        expected_trains = [
            {
                "number_train": "456",
                "departure_point": "Казань",
                "time_departure": "14:00",
                "destination": "Москва"
            },
            {
                "number_train": "789",
                "departure_point": "Сочи",
                "time_departure": "16:00",
                "destination": "Москва"
            }
        ]
        self.assertEqual(len(selected_trains), len(expected_trains))
        for train in expected_trains:
            self.assertIn(train, selected_trains)


if __name__ == "__main__":
    unittest.main(testRunner=CustomTestRunner)
```

Результаты работы программы теста, успешное завершение программы (рис. 3).

```
C:\Users\Andrey\Desktop\Анализ_данных\8_лаба\Analysis_data_laboratory_work_8\individual>python tests.py -v
test_add_train (__main__.TrainManagementTest.test_add_train)
Тест добавления поездов ... ok
test_add_train (__main__.TrainManagementTest.test_add_train)
Тест добавления поездов ... ok
test_select_all (__main__.TrainManagementTest.test_select_all)
Проверка выбора всех поездов. ... ok
test_select_all (__main__.TrainManagementTest.test_select_all)
Проверка выбора всех поездов. ... ok
test_select_by_destination (__main__.TrainManagementTest.test_select_by_destination)
Тестирование поездов по пункту назначения ... ok
test_select_by_destination (__main__.TrainManagementTest.test_select_by_destination)
Тестирование поездов по пункту назначения ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.057s

OK

C:\Users\Andrey\Desktop\Анализ_данных\8_лаба\Analysis_data_laboratory_work_8\individual>
```

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