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»
Вариант 13

Выполнил:
Иващенко Олег Андреевич
2 курс, группа ИВТ-б-о-22-1,
09.03.02 «Информационные и вычислительные машины»,
направленность (профиль)
«Программное обеспечение средств вычислительной техники и автоматизированных систем»

_____
(подпись)

Руководитель практики:
Воронкин Роман Александрович,
доцент кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: «Тестирование в Python [unittest]»

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

Порядок выполнения работы

Листинг 1.1 – Пример 1 (модуль calc.py)

```
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
```

Листинг 1.2 – Пример 1 (модуль test_calc.py)

```
import calc

def test_add():
    if calc.add(1, 2) == 3:
        print("Test add(a, b) is OK")
    else:
        print("Test add(a, b) is Fail")

def test_sub():
    if calc.sub(4, 2) == 2:
        print("Test sub(a, b) is OK")
    else:
        print("Test sub(a, b) is Fail")

def test_mul():
    if calc.mul(2, 5) == 10:
        print("Test mul(a, b) is OK")
    else:
        print("Test mul(a, b) is Fail")

def test_div():
    if calc.div(8, 4) == 2:
        print("Test div(a, b) is OK")
    else:
        print("Test div(a, b) is Fail")

test_add()
test_sub()
test_mul()
test_div()
```

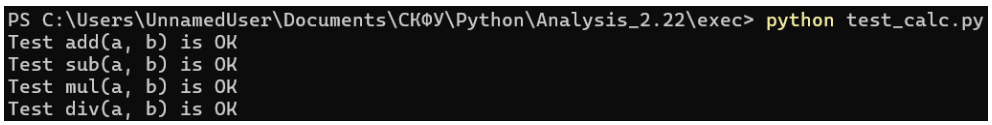

Рисунок 1.1 – Запуск теста

Листинг 1.3 – Пример 2 (unittest)

```
import unittest
import calc

class CalcTest(unittest.TestCase):
    def test_add(self):
        self.assertEqual(calc.add(1, 2), 3)

    def test_sub(self):
        self.assertEqual(calc.sub(4, 2), 2)

    def test_mul(self):
        self.assertEqual(calc.mul(2, 5), 10)

    def test_div(self):
        self.assertEqual(calc.div(8, 4), 2)

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

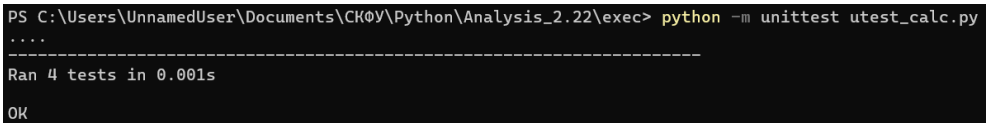

Рисунок 1.2 – Запуск теста unittest

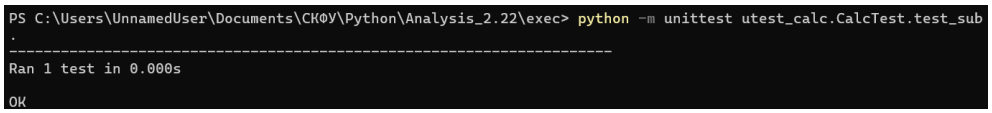

Рисунок 1.3 – Тест отдельного метода модуля (unittest)

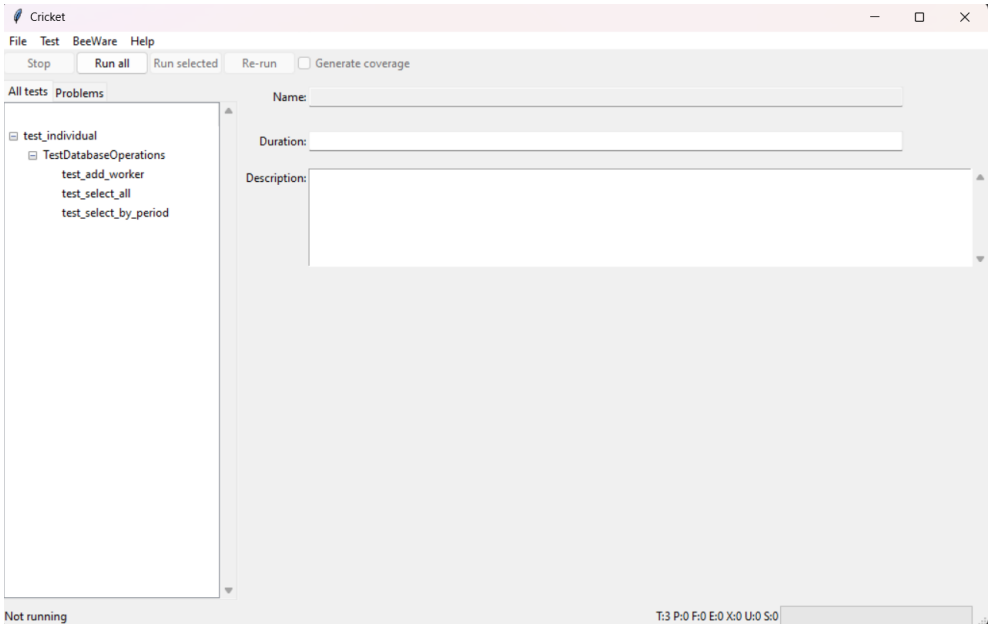

Рисунок 1.4 – Запуск Cricket

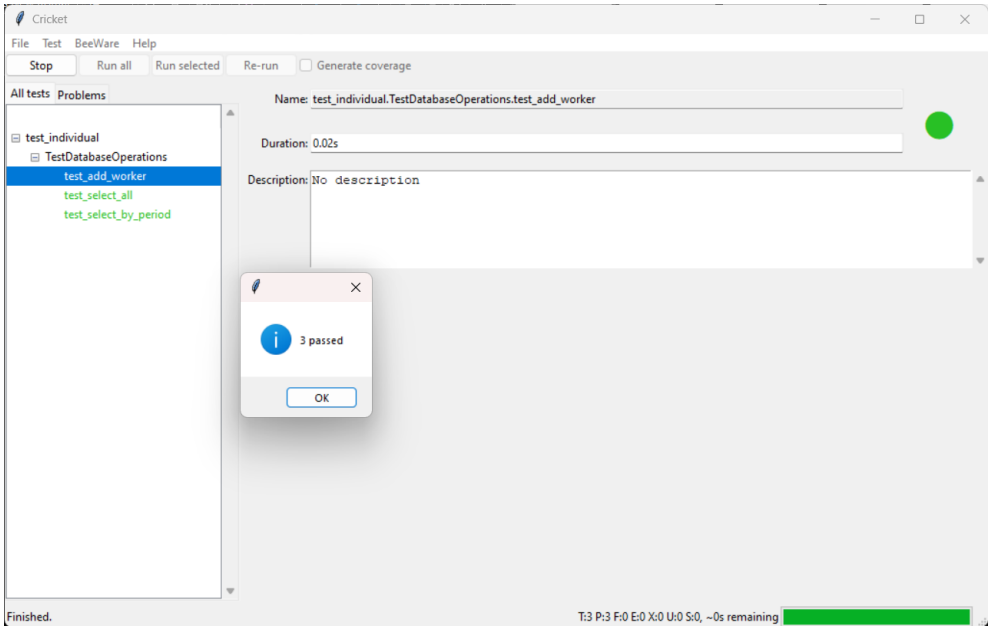

Рисунок 1.5 – Тестирование с помощью Cricket

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

Листинг 2.1 – Код individual.py

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


"""
```

```
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.
"""

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


def print_help():
    """
    Функция вывода доступных пользователю команд
    """

    print("list - вывод всех добавленных записей")
    print("add - добавление новых записей")
    print("find - найти запись по фамилии")
    print("exit - завершение работы программы")


def add_worker(database_path: Path, name: str, phone: str, year: int) -> None:
    """
    Функция добавления новой записи, возвращает запись
    """
    connection = sqlite3.connect(database_path)
    cursor = connection.cursor()

    cursor.execute(
        """
        INSERT INTO workers (worker_name, phone_number, worker_year)
        VALUES (?, ?, ?)
        """,
        (name, phone, year)
    )

    connection.commit()
    connection.close()


def print_list(staff: t.List[t.Dict[str, t.Any]]) -> None:
```

```
    """
    Функция выводит на экран список всех существующих записей
    """
    if staff:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )
        print(line)

        print(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                "№",
                "Ф.И.О.",
                "Номер телефона",
                "Год"
            )
        )
        print(line)

        for idx, worker in enumerate(staff, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                    idx,
                    worker.get('name', ''),
                    worker.get('phone_number', ''),
                    worker.get('year', 0)
                )
            )
            print(line)

    else:
        print("Список работников пуст.")


def create_db(database_path: Path) -> None:
    """
    Создание базы данных
    """
    connection = sqlite3.connect(database_path)
```

```
    cursor = connection.cursor()

    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS workers (
            worker_id INTEGER PRIMARY KEY AUTOINCREMENT,
            worker_name TEXT NOT NULL,
            phone_number TEXT NOT NULL,
            worker_year INTEGER NOT NULL
        )
        """
    )

    connection.close()


def select_all(database_path: Path) -> t.List[t.Dict[str, t.Any]]:
    """
    Выбор всех записей из базы данных
    """
    connection = sqlite3.connect(database_path)
    cursor = connection.cursor()

    cursor.execute(
        """
        SELECT workers.worker_name, workers.phone_number, workers.worker_year
        FROM workers
        """
    )
    rows = cursor.fetchall()

    connection.close()
    return [
        {
            "name": row[0],
            "phone_number": row[1],
            "year": row[2],
        }
        for row in rows
    ]
```

```
def select_by_period(
    database_path: Path, period: int
) -> t.List[t.Dict[str, t.Any]]:
    """
    Выборка по периоду
    """
    connection = sqlite3.connect(database_path)
    cursor = connection.cursor()

    cursor.execute(
        """
        SELECT workers.worker_name, workers.phone_number, workers.worker_year
        FROM workers
        WHERE (strftime('%Y', date('now')) - workers.worker_year) >= ?
        """,
        (period,)
    )
    rows = cursor.fetchall()

    connection.close()
    return [
        {
            "name": row[0],
            "phone_number": row[1],
            "year": row[2],
        }
        for row in rows
    ]



def main(command_line=None):
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "--db",
        action="store",
        required=False,
        default=str(Path.home() / "workers.db"),
        help="Название файла базы даанных"
    )

    parser = argparse.ArgumentParser("workers")
    parser.add_argument(
```

```
    "--version",
    action="version",
    version="%(prog)s 0.1.0"
)

subparsers = parser.add_subparsers(dest="command")

add = subparsers.add_parser(
    "add",
    parents=[file_parser],
    help="Add a new worker"
)
add.add_argument(
    "-n",
    "--name",
    action="store",
    required=True,
    help="Имя работника"
)
add.add_argument(
    "-p",
    "--phone",
    action="store",
    help="Номер телефона работника"
)
add.add_argument(
    "-y",
    "--year",
    action="store",
    required=True,
    help="Дата нанятия"
)

_ = subparsers.add_parser(
    "display",
    parents=[file_parser],
    help="Вывести на экран всех работников"
)

select = subparsers.add_parser(
    "select",
    parents=[file_parser],
```

```
        help="Выборка работников"
    )
    select.add_argument(
        "-p",
        "--period",
        action="store",
        type=int,
        required=True,
        help="Требуемый период"
    )

    args = parser.parse_args(command_line)

    db_path = Path(args.db)
    create_db(db_path)

    if args.command == "add":
        add_worker(db_path, args.name, args.phone, args.year)

    elif args.command == "display":
        print_list(select_all(db_path))

    elif args.command == "select":
        print_list(select_by_period(db_path, args.period))
        pass


if __name__ == "__main__":
    """
    Основная программа
    """
    main()
```

Листинг 2.2 – Код test_individual.py

```
import unittest
from pathlib import Path
import tempfile
import os
from individual import create_db, add_worker, select_all, select_by_period


«»»
```

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


class TestDatabaseOperations(unittest.TestCase):
    def    eard(self):
        self.db_fd, self.db_path = tempfile.mkstemp()
        create_db(Path(self.db_path))

    def    eardown(self):
        os.close(self.db_fd)
        os.unlink(self.db_path)

    def test_add_worker(self):
        add_worker(Path(self.db_path), “Петров Петр”, “79998886452”, 2000)
        workers = select_all(Path(self.db_path))
        self.assertEqual(len(workers), 1)
        self.assertEqual(workers[0][‘name’], “Петров Петр”)
        self.assertEqual(workers[0][‘phone_number’], “79998886452”)
        self.assertEqual(workers[0][‘year’], 2000)

    def test_select_all(self):
        add_worker(Path(self.db_path), “Петров Петр”, “79998886452”, 2000)
        add_worker(Path(self.db_path), “Иванов Иван”, “79995554291”, 2015)
        workers = select_all(Path(self.db_path))
        self.assertEqual(len(workers), 2)

    def test_select_by_period(self):
        add_worker(Path(self.db_path), “Петров Петр”, “79998886452”, 2000)
        add_worker(Path(self.db_path), “Иванов Иван”, “79995554291”, 2015)
        workers = select_by_period(Path(self.db_path), 10)
        self.assertEqual(len(workers), 1)
        self.assertEqual(workers[0][‘name’], “Петров Петр”)


if __name__ == ‘__main__’:
    unittest.main()
```

```
PS C:\Users\UnnamedUser\Documents\СКФУ\Python\Analysis_2.22\exec> python -m unittest test_individual.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.082s

OK
```

Рисунок 2.1 – Тестирование individual.py (unittest)

```
PS C:\Users\UnnamedUser\Documents\СКФУ\Python\Analysis_2.22\exec> python -m unittest -v test_individual.py
test_add_worker (test_individual.TestDatabaseOperations.test_add_worker) ... ok
test_select_all (test_individual.TestDatabaseOperations.test_select_all) ... ok
test_select_by_period (test_individual.TestDatabaseOperations.test_select_by_period) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.079s

OK
```

Рисунок 2.2 – Подробное тестирование individual.py (unittest)

Контрольные вопросы

1. Для чего используется автономное тестирование?

Автономное тестирование используется для проверки отдельных частей программы, таких как функции и классы, изолированно от остальной системы, чтобы гарантировать их правильную работу.

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

Наиболее распространённые фреймворки для автономного тестирования в Python – unittest, pytest и nose.

3. Какие существуют основные структурные единицы модуля unittest?

Основные структурные единицы модуля unittest:

- TestCase – класс для создания тестов;
- TestSuite – класс для объединения нескольких тестов;
- TestLoader – класс для загрузки классов;
- TextTestRunner – класс для запуска тестов;
- TestResult – класс для хранения результатов тестов.

4. Какие существуют способы запуска тестов unittest?

Unittest можно запустить из командной строки с помощью команды python -m unittest <названи_файла> или с помощью программного вызова через unittest.main().

5. Каково назначение класса TestCase?

Класс TestCase используется для создания тестов, определяя методы, начинающиеся с test, которые содержат проверяемые условия.

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

Методы класса TestCase, выполняемые при запуске и завершении работы тестов – setup() (выполняется перед каждым тестом) и tearDown() (выполняется после каждого теста).

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

Методы класса TestCase для проверки условий и генерации ошибок:

- assertEqual();
- assertTrue();
- assertFalse();
- assertRaises();
- assertIn();
- assertNotIn();
- assertIs();
- assertIsNot();
- assertIsNone();
- assertIsNotNone().

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

Методы класса TestCase для собора информации о тесте:

- id() – возвращает уникальный идентификатор теста;
- shortDescription() – возвращает краткое описание теста (если задано в виде строки документации).

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

- Класс TestSuite используется для объединения нескольких тестов в одну группу для их совместного запуска. Загрузка тестов

осуществляется с помощью методов addTest(), addTests() и с помощью класса TestLoader.

10. Каково назначение класса TestResult?

Класс TestResult используется для хранения и отображения результатов тестирования, включая информацию о пройденных, проваленных и пропущенных тестах.

11. Для чего может понадобиться пропуск отдельных тестов?

Пропуск отдельных тестов может понадобиться, если тест ещё не реализован, зависит от внешних условий или временно неактуален.

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

Безусловный пропуск тестов можно выполнить с помощью декоратора @unittest.skip(«причина»).

Условный пропуск тестов можно выполнить с помощью декораторов @unittest.skipIf(condition, «причина») и @unittest.skipUnless(condition, «причина»).

Пропуск класса тестов можно выполнить с помощью декорирования всего класса с помощью @unittest.skip(«причина»).

Выводы: В процессе выполнения лабораторной работы были изучены способы тестирования программ/модулей с помощью различных инструментов, начиная от встроенных в Python и заканчивая дополнительными программными обеспечениями для проведения тестирования модуля с помощью GUI. Были проработаны примеры лабораторной работы, а также выполнено индивидуальное задание.