

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ**

ФЕДЕРАЦИИ

**Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Кафедра инфокоммуникаций

«Тестирование в Python [unittest]»

Отчет по лабораторной работе № 2.22

по дисциплине «Основы программной инженерии»

Выполнил студент группы ПИЖ-б-о-21-1

Кучеренко С. Ю. _____ « » 2023г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

Проверил Воронкин Р.А. _____

(подпись)

Ставрополь 2023

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

Ход работы:

1. Изучить теоретический материал работы.
2. Проработайте примеры лабораторной работы.
3. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.
4. Выполните клонирование созданного репозитория.
5. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.
6. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.
7. Создайте проект PyCharm в папке репозитория.
8. Выполните индивидуальное задание. Приведите в отчете скриншоты работы программы решения индивидуального задания.

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

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

import unittest
import sqlite3
from pathlib import Path
from ind import create_db, add_way, select_all, find_ways

class HumanTests(unittest.TestCase):
    # Метод действует на уровне класса, т.е. выполняется
    # перед запуском тестов класса.
    @classmethod
    def setUpClass(cls):
        """Set up for class"""

        print("setUpClass")
        print("=====")

    # Запускается после выполнения всех методов класса
    @classmethod
    def tearDownClass(cls):
        """Tear down for class"""
        print("=====")
```

```

print("tearDownClass")

# Метод вызывается перед запуском теста. Как правило,
# используется для подготовки окружения для теста.
def setUp(self):
    """Set up for test"""
    print(f"\nSet up for [{self.shortDescription()}]")

# Метод вызывается после завершения работы теста.
# Используется для "приборки" за тестом.
def tearDown(self):
    """Tear down for test"""
    print(f"Tear down for [{self.shortDescription()}]")

def test_create_db(self):
    """ Checking the database creation. """
    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_human(self):
    """ Checking the addition. """
    database_path = "test.db"
    create_db(database_path)
    add_way(database_path, "ppp", "mmm", 20)
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    cursor.execute(
        """
        SELECT * FROM waypoints
        """
    )
    row = cursor.fetchone()
    self.assertEqual(row, (1, "ppp", 1, "mmm"))
    conn.close()
    Path(database_path).unlink()

def test_select_all(self):
    """ Checking the selection all """
    database_path = "test.db"
    create_db(database_path)
    add_way(database_path, "ppp", "mmm", 33)
    add_way(database_path, "www", "nnn", 44)

    comparison_output = [
        {"start": "ppp", "finish": 'mmm', "num": 33},
        {"start": "www", "finish": 'nnn', "num": 44},
    ]
    self.assertEqual(select_all(database_path), comparison_output)
    Path(database_path).unlink()

def test_find_ways(self):
    """ Checking the selection by routes number """
    database_path = "test.db"
    create_db(database_path)
    add_way(database_path, "ppp", "mmm", 33)
    add_way(database_path, "www", "nnn", 44)
    comparison_output = [
        {"start": "ppp", "finish": "mmm", "num": 33},
    ]

```

```
self.assertEqual(find_ways(database_path, 33), comparison_output)
Path(database_path).unlink()
```

```
Tear down for [Checking the database creation.]
ok
test_select_all (tests_routes.HumanTests.test_select_all)
Checking the selection all ...
Set up for [Checking the selection all]
Tear down for [Checking the selection all]
ok
test_select_zz (tests_routes.HumanTests.test_select_zz)
Checking the selection by routes number ...
Set up for [Checking the selection by routes number]
Tear down for [Checking the selection by routes number]
ok
=====
tearDownClass

-----

Ran 4 tests in 0.008s

OK
(venv) (base) svetik@MacBook-Air-Svetik PyCharm %
```

Рисунок 1 – Результат работы программы

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

import unittest
import tests_routes

prodTestSuite = unittest.TestSuite()
prodTestSuite.addTest(unittest.makeSuite(tests_routes.RoutesTests))
runner = unittest.TextTestRunner(verbosity=2)
runner.run(prodTestSuite)
```

```
=====
tearDownClass

-----

Ran 4 tests in 0.009s

OK
(venv) (base) svetik@MacBook-Air-Svetik PyCharm %
```

Рисунок 2 – Результат работы программы

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

from pathlib import Path
```

```

import sqlite3
from ind import create_db, add_way, select_all, find_ways

class TestsRoutes:
    """
    Program test for a list of routes
    """

    def test_create_db(self):
        """
        Checking the database creation.
        """
        database_path = "test.db"
        if Path(database_path).exists():
            Path(database_path).unlink()

        create_db(database_path)
        assert Path(database_path).is_file()
        Path(database_path).unlink()

    def test_add_way(self):
        """
        Checking the addition.
        """
        database_path = "test.db"
        create_db(database_path)
        add_way(database_path, "ppp", "mmm", 20)
        conn = sqlite3.connect(database_path)
        cursor = conn.cursor()
        cursor.execute(
            """
            SELECT * FROM waypoints
            """
        )
        row = cursor.fetchone()
        assert row == (1, "ppp", 1, "mmm")
        conn.close()
        Path(database_path).unlink()

    def test_select_all(self):
        """
        Checking the selection all.
        """
        database_path = "test.db"
        create_db(database_path)
        add_way(database_path, "ppp", "mmm", 33)
        add_way(database_path, "www", "nnn", 44)

        comparison_output = [
            {"start": "ppp", "finish": 'mmm', "num": 33},
            {"start": "www", "finish": 'nnn', "num": 44},
        ]
        assert select_all(database_path) == comparison_output
        Path(database_path).unlink()

    def test_find_ways(self):
        """
        Checking the selection by routes number
        """
        database_path = "test.db"

```

```

create_db(database_path)
add_way(database_path, "ppp", "mmm", 33)
add_way(database_path, "www", "nnn", 44)

comparison_output = [
    {"start": "ppp", "finish": "mmm", "num": 33},
]
assert find_ways(database_path, 33) == comparison_output
Path(database_path).unlink()

```

```

platform darwin -- Python 3.9.13, pytest-7.1.2, pluggy-1.0.0
rootdir: /Users/svetik/Desktop/OPI/OPI_LR_2.22/PyCharm
plugins: anyio-3.5.0
collected 8 items

tests/test_pytest.py ....
tests/test_routes.py ....

===== 8 passed in 0.07s =====
(base) svetik@MacBook-Air-Svetik PyCharm %

```

Рисунок 3 – Результат работы программы

9. Зафиксируйте сделанные изменения в репозитории.
10. Выполните слияние ветки для разработки с веткой main (master).
11. Отправьте сделанные изменения на сервер GitHub.

Контрольные вопросы:

1. Для чего используется автономное тестирование?

Автономное тестирование используется для автоматизации процесса проверки качества программного обеспечения. Вместо ручного выполнения тестовых сценариев автономное тестирование позволяет создавать и запускать тесты с использованием специальных инструментов или программных скриптов.

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

В мире Python существуют три framework'а, которые получили наибольшее распространение: unittest, nose, pytest.

3. Какие существуют основные структурные единицы модуля unittest?

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 – 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 – 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 – 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?

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.

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.

setUpClass() – Метод действует на уровне класса, т.е. выполняется перед запуском тестов класса. При этом синтаксис требует наличие декоратора @classmethod.

tearDownClass() – Запускается после выполнения всех методов класса, требует наличия декоратора @classmethod.

skipTest(reason) – Данный метод может быть использован для пропуска теста, если это необходимо.

7. Какие методы класса TestCase используются для проверки условий и генерации ошибок? assertEquals(a, b):

Этот метод проверяет, равны ли значения a и b. Если значения не равны, то тест считается неудачным, и генерируется ошибка.

assertNotEqual(a, b):

Этот метод проверяет, не равны ли значения a и b. Если значения равны, то тест считается неудачным, и генерируется ошибка.

assertTrue(x):

Этот метод проверяет, является ли значение x истинным (равным True). Если значение x не является истинным, то тест считается неудачным, и генерируется ошибка. assertFalse(x):

Этот метод проверяет, является ли значение x ложным (равным False). Если значение x не является ложным, то тест считается неудачным, и генерируется ошибка.

assertIs(a, b):

Этот метод проверяет, являются ли объекты `a` и `b` одним и тем же объектом (ссылаются ли они на одно и то же место в памяти). Если объекты не являются одним и тем же объектом, то тест считается неудачным, и генерируется ошибка. `assertIsNot(a, b)`:

Этот метод проверяет, не являются ли объекты `a` и `b` одним и тем же объектом. Если объекты являются одним и тем же объектом, то тест считается неудачным, и генерируется ошибка.

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?

Пропуск отдельных тестов может быть полезным для временного отключения тестов, неподдерживаемых платформ или конфигураций, тестов в разработке или устаревших тестов.

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

Для пропуска теста воспользуемся декоратором, который пишется перед тестом: `@unittest.skip(reason)`

Для условного пропуска тестов применяются следующие декораторы: `@unittest.skipIf(condition, reason)` (Тест будет пропущен, если условие (condition) истинно) и `@unittest.skipUnless(condition, reason)` (Тест будет пропущен если, условие (condition) не истинно).

Для пропуска классов используется декоратор `@unittest.skip(reason)`

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

Алгоритм проведения тестирования с использованием PyCharm:

1. Настройка окружения тестирования.

Установите и настройте PyCharm для проекта, включая настройку интерпретатора Python, настройку путей к исходным файлам и тестовым файлам, а также настройку модуля unittest.

2. Создание тестовых файлов.

Создайте файлы с тестами в проекте. Обычно тестовые файлы содержат классы наследующие `unittest.TestCase` и содержащие тестовые методы.

3. Определение тестовых методов.

Определите тестовые методы внутри классов, используя методы `assert` для проверки ожидаемых результатов. Каждый тестовый метод должен начинаться с префикса "test".

4. Настройка конфигурации тестирования.

В PyCharm выберите конфигурацию для запуска тестов. Вы можете выбрать определенный тестовый файл, папку с тестами или использовать общую конфигурацию для запуска всех тестов в проекте.

5. Запуск тестов.

Запустите тесты, выбрав соответствующую конфигурацию тестирования. PyCharm выполнит все тестовые методы и выведет результаты выполнения в специальной панели или в консоли.

6. Анализ результатов тестирования.

7. Исправление проблем в коде или тестах.

8. Повторение итераций до достижения требуемого поведения программы.