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 «Основы кроссплатформенного программирования»**

|  |
| --- |
| Выполнил:  Быковская Стефания Станиславовна  2 курс, группа ИТС-б-о-22-1,  11.03.02 «Инфокоммуникационные технологии и системы связи»,  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.** Создали общедоступный репозиторий на GitHub, в котором использована лицензий MIT и язык программирования Python, также добавила файл .gitignore с необходимыми правилами. Клонировали свой репозиторий на свой компьютер. Организовали свой репозиторий в соответствие с моделью ветвления git-flow, появилась новая ветка develop в которой буду выполнять дальнейшие задачи.

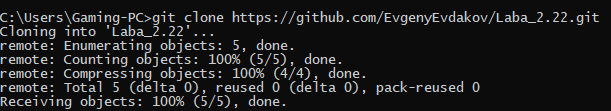


Рисунок 1. Клонирование репозитория

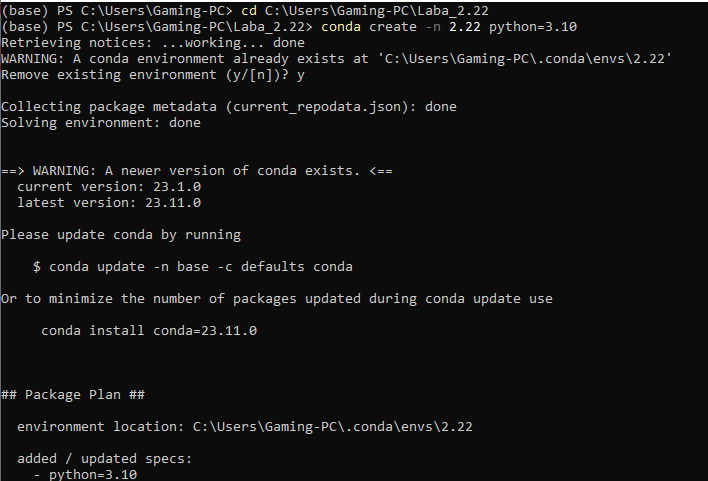
**Задание 2.** Создали виртуальное окружение conda и активировала его, также установила необходимые пакеты isort, black, flake8.

Рисунок 2. Создание виртуального окружения

**Задание 3.** Создали проект PyCharm в папке репозитория. Приступила к работе с примером. Добавили новый файл primer1.py.

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

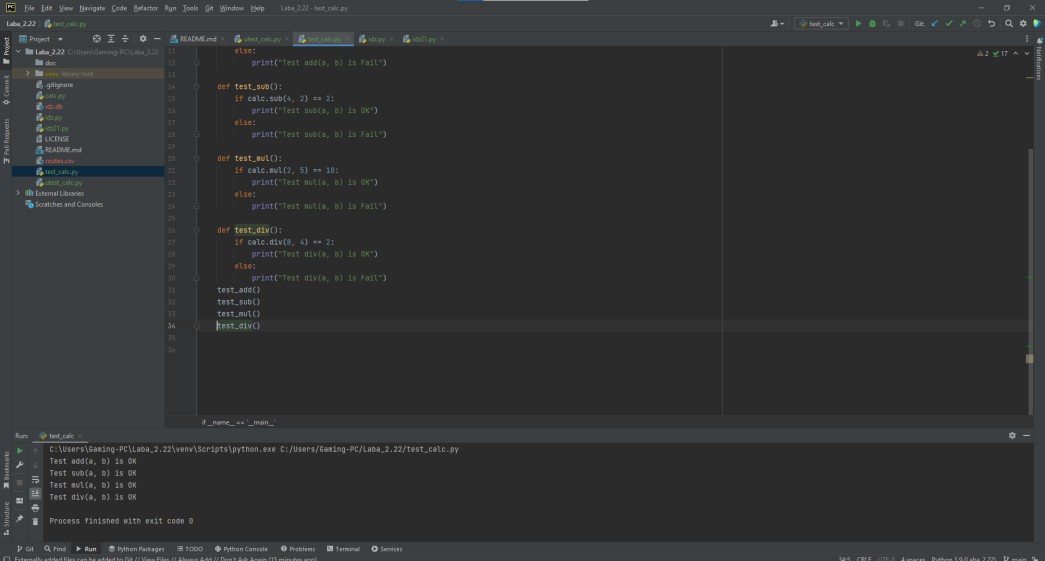


Рисунок 3. Выполнение первого примера

# Индивидуальное задание

# Вариант 2

Создали новый файл под названием idz.py.

Условие задания:Для индивидуального задания лабораторной работы

2.22 добавьте тесты с использованием модуля unittest, проверяющие операции по работе с базой данных.

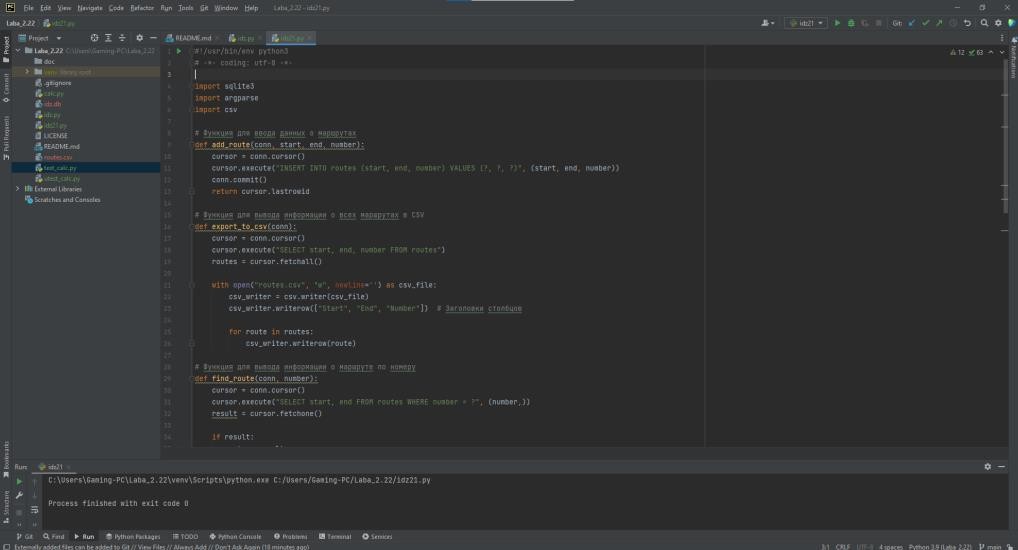


Рисунок 4. Основной код

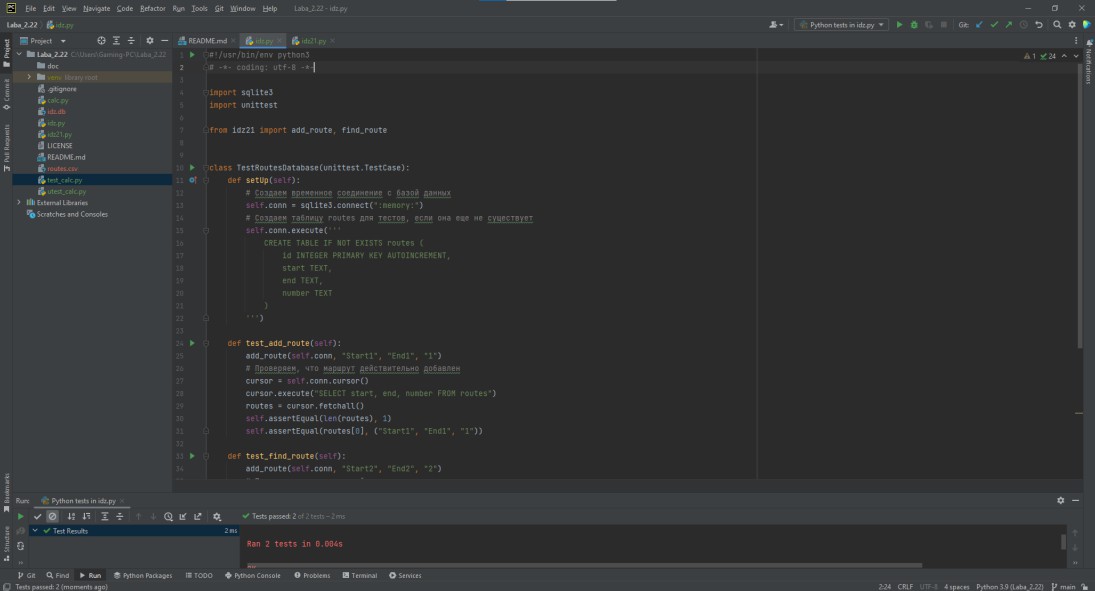


Рисунок 5. Код для индивидуального задания

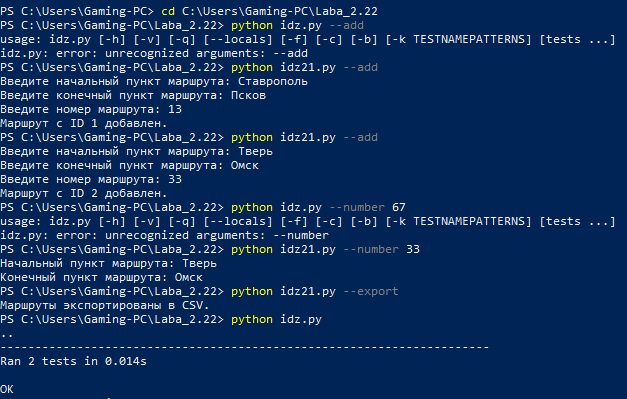


Рисунок 6. Результат выполнения

# Задание 4.

После выполнения работы на ветке develop, слила ее с веткой main и отправила изменения на удаленный сервер. Создала файл envirement.yml и деактивировала виртуальное окружение.

Рисунок 7. Деактивация ВО

Ссылка: https://github.com/stefa-b/Phyton-22.2.git

# 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.

# 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.

# 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.

# 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.

# 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.

# 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.

# 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.

# 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.

# 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.

# 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.

# 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.

# 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.

# 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
3. Создать файл с тестами (обычно файл с префиксом test\_).
4. Определить классы тестов, унаследованные от unittest.TestCase.
5. Написать методы тестов внутри классов, используя методы assert для проверки условий.
   * Шаг 3: Запуск тестов
6. Открыть файл с тестами.
7. Нажать правой кнопкой мыши и выбрать "Run 'pytest in test\_file.py'"
8. Посмотреть результаты выполнения тестов в окне вывода.
   * Шаг 4: Анализ результатов
9. После выполнения тестов, PyCharm предоставит подробные результаты в специальной вкладке "Run" внизу экрана.
10. Анализировать результаты успешных и неуспешных тестов, и, при необходимости, вносить исправления в код.

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