

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 «Программирование на языке Python»

Выполнил:
Хачатрян Владимир Владимирович
2 курс, группа ИТС-б-о-22-1,
11.03.02 «Инфокоммуникационные
технологии и системы связи»,
направленность (профиль)
«Инфокоммуникационные системы и
сети», очная форма обучения

(подпись)

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

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

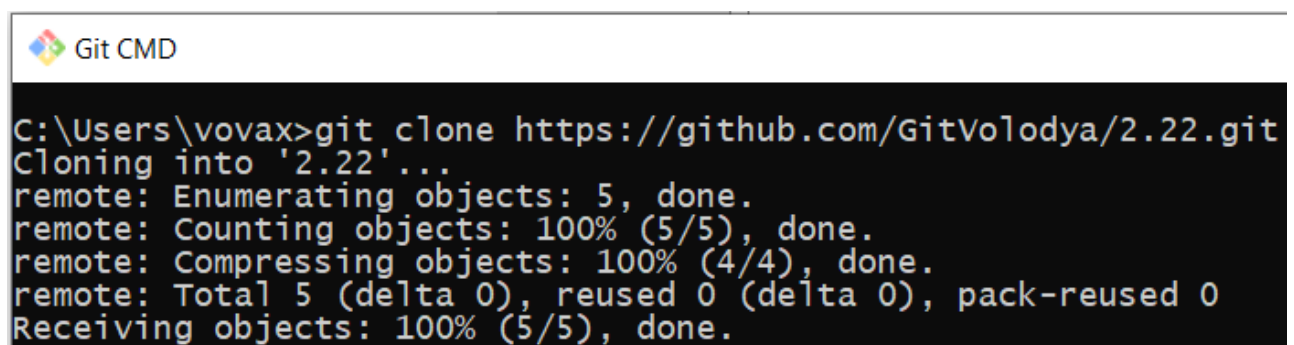
Ставрополь, 2023 г.

Тема: тестирование в Python [unittest]

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

Выполнение:

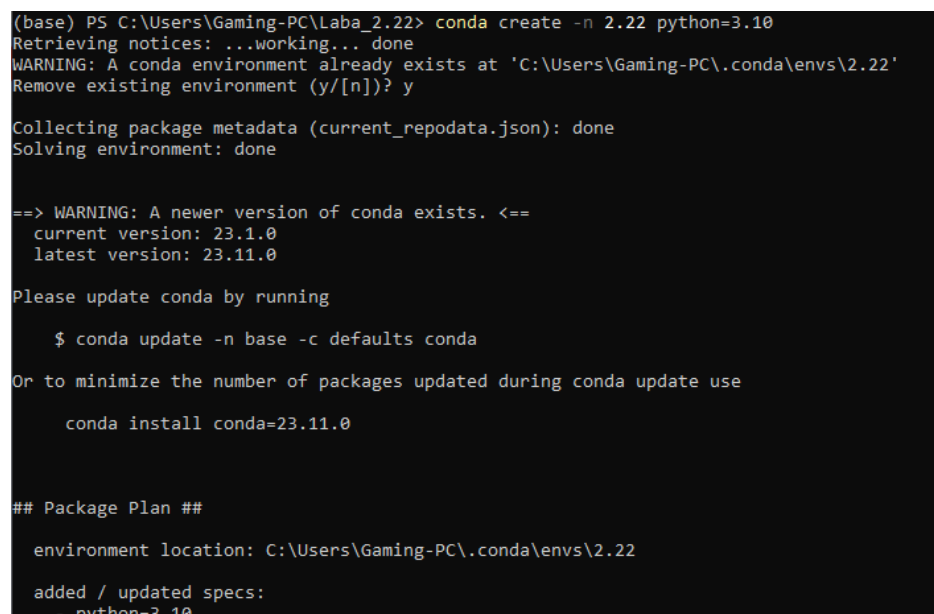
Создал общедоступный репозиторий на GitHub, в котором использована лицензий MIT и язык программирования Python, также добавил файл .gitignore с необходимыми правилами. Клонировал свой репозиторий на свой компьютер. Организовал свой репозиторий в соответствии с моделью ветвления git-flow, появилась новая ветка develop в которой буду выполнять дальнейшие задачи.



```
Git CMD
C:\Users\vovax>git clone https://github.com/GitVolodya/2.22.git
Cloning into '2.22'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
```

Рисунок 1. Клонирование репозитория

Создал виртуальное окружение conda и активировал его, также установил необходимые пакеты isort, black, flake8.



```
(base) PS C:\Users\Gaming-PC\Laba_2.22> conda create -n 2.22 python=3.10
Retrieving notices: ...working... done
WARNING: A conda environment already exists at 'C:\Users\Gaming-PC\.conda\envs\2.22'
Remove existing environment (y/[n])? y

Collecting package metadata (current_repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 23.11.0
  latest version: 23.11.0

Please update conda by running

    $ conda update -n base -c defaults conda

Or to minimize the number of packages updated during conda update use

    conda install conda=23.11.0

## Package Plan ##

  environment location: C:\Users\Gaming-PC\.conda\envs\2.22
  added / updated specs:
    - python=3.10
```

Рисунок 2. Создание виртуального окружения

Создал проект PyCharm в папке репозитория. Приступил к работе с примером. Добавил новый файл `primer1.py`.

Условие: написать код, который будет проверять правильность работы файла `calc.py`

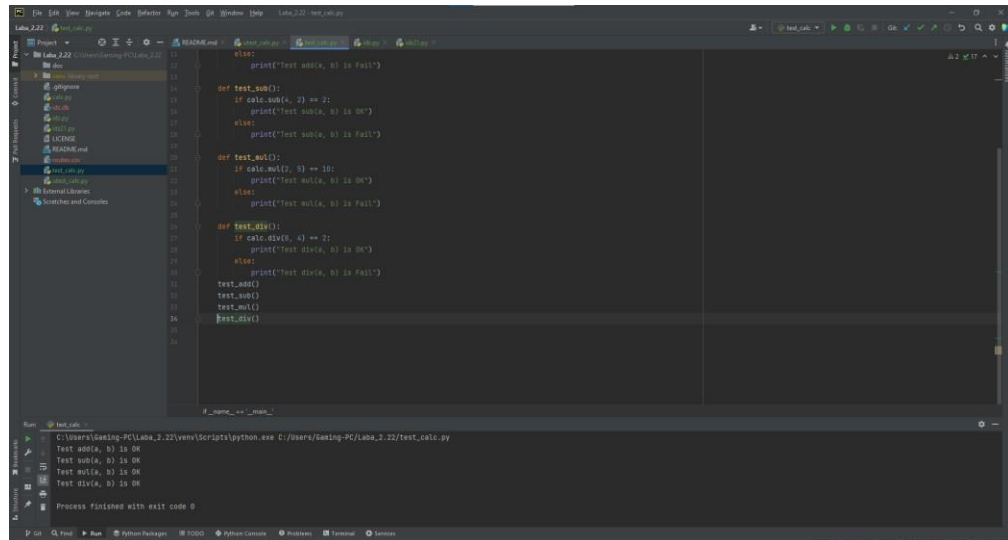


Рисунок 3. Выполнение примера

Индивидуальное задание

Создал новый файл под названием `idz.py`. Для индивидуального задания лабораторной работы 2.21 добавьте тесты с использованием модуля `unittest`, проверяющие операции по работе с базой данных.

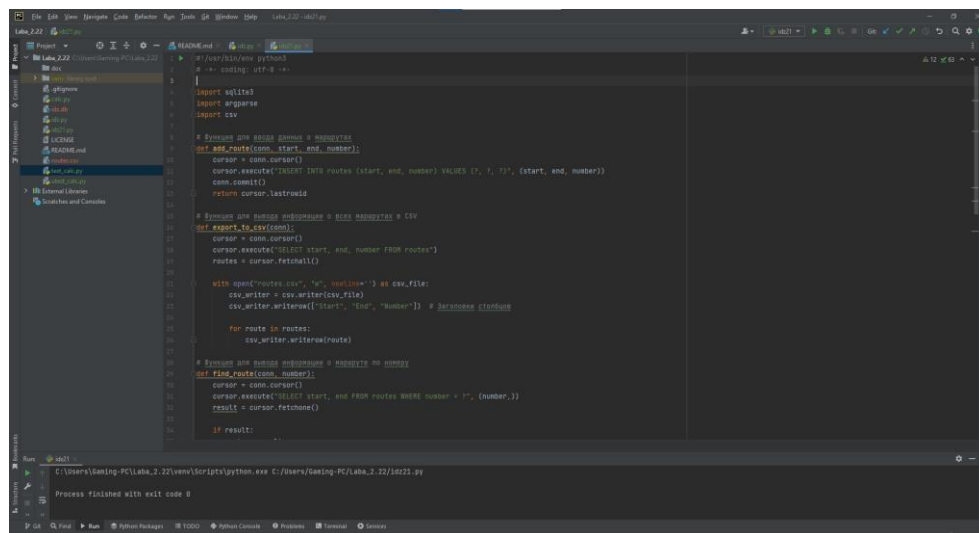


Рисунок 4. Код

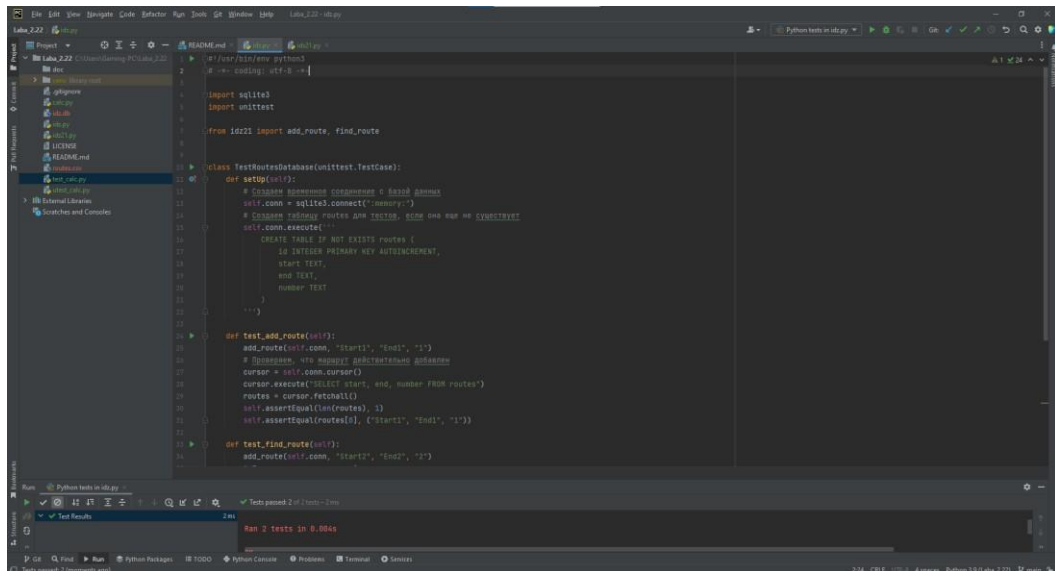


Рисунок 5. Код для индивидуального задания

```

PS C:\Users\Gaming-PC> cd C:\Users\Gaming-PC\Laba_2.22
PS C:\Users\Gaming-PC\Laba_2.22> python idz.py --add
usage: idz.py [-h] [-v] [-q] [--locals] [-f] [-c] [-b] [-k TESTNAMEPATTERNS] [tests ...]
idz.py: error: unrecognized arguments: --add
PS C:\Users\Gaming-PC\Laba_2.22> python idz21.py --add
Введите начальный пункт маршрута: Ставрополь
Введите конечный пункт маршрута: Псков
Введите номер маршрута: 13
Маршрут с ID 1 добавлен.
PS C:\Users\Gaming-PC\Laba_2.22> python idz21.py --add
Введите начальный пункт маршрута: Тверь
Введите конечный пункт маршрута: Омск
Введите номер маршрута: 33
Маршрут с ID 2 добавлен.
PS C:\Users\Gaming-PC\Laba_2.22> python idz.py --number 67
usage: idz.py [-h] [-v] [-q] [--locals] [-f] [-c] [-b] [-k TESTNAMEPATTERNS] [tests ...]
idz.py: error: unrecognized arguments: --number
PS C:\Users\Gaming-PC\Laba_2.22> python idz21.py --number 33
Начальный пункт маршрута: Тверь
Конечный пункт маршрута: Омск
PS C:\Users\Gaming-PC\Laba_2.22> python idz21.py --export
Маршруты экспортированы в CSV.
PS C:\Users\Gaming-PC\Laba_2.22> python idz.py
..
-----
Ran 2 tests in 0.014s
OK

```

Рисунок 6. Результат

Отправил изменения на удаленный сервер. Создал файл envirement.yml и деактивировал виртуальное окружение.

Ссылка на репозиторий: <https://github.com/GitVolodya/2.22.git>

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 assertEquals, 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 работы приобрел навыки написания автоматизированных тестов на языке программирования Python версии 3.x.