МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное

образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

Отчет по лабораторной работе №2.22

Работа с переменными окружения в Python3

по дисциплине «Тестирование в Python [unittest]»

Выполнил студент группы ИВТ-б-о-20-1

Хашиев Х.М. « »_____20__г.

Подпись студента_____

Работа защищена « »_____20__г.

Проверил Воронкин Р.А. _____

(подпись

Ставрополь 2022

Цель работы: приобретение навыков с тестированием с помощью языка программирования Python версии 3.x.

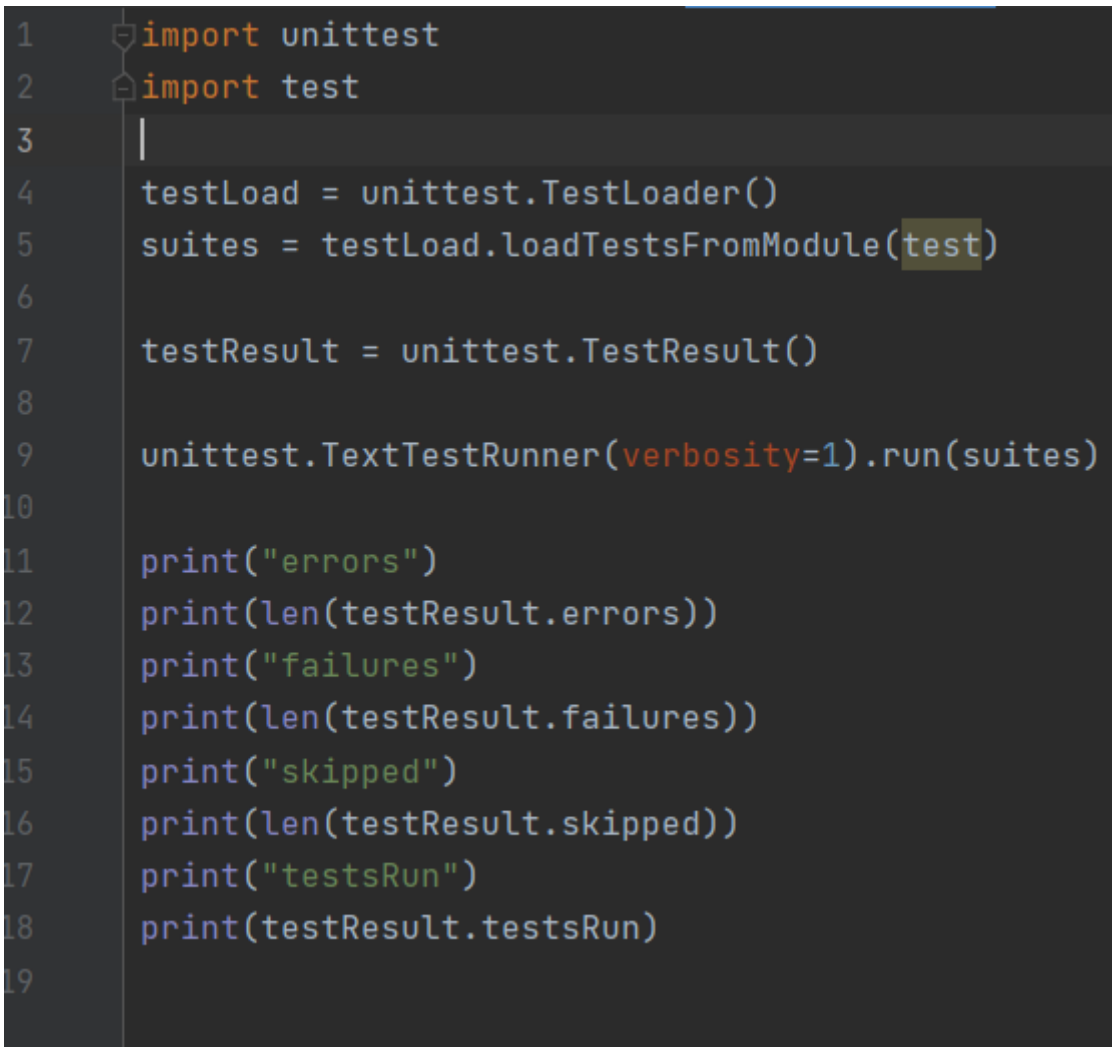
Ход работы: Примеры

https://github.com/Mirror-Shard/L2.22

1. Изучил теоретический материал и приступил к выполнению примеров:

```python
import unittest
import calc


class CalcBasicTests(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)


class CalcExTests(unittest.TestCase):
    def test_sqrt(self):
        self.assertEqual(calc.sqrt(4), 2)

    def test_pow(self):
        self.assertEqual(calc.pow(3, 3), 27)
```

Рисунок 1 – Тестирование calc



```
import unittest
import test

testLoad = unittest.TestLoader()
suites = testLoad.loadTestsFromModule(test)

testResult = unittest.TestResult()

unittest.TextTestRunner(verbosity=1).run(suites)

print("errors")
print(len(testResult.errors))
print("failures")
print(len(testResult.failures))
print("skipped")
print(len(testResult.skipped))
print("testsRun")
print(testResult.testsRun)
```

Рисунок 2 – Test runner для файла calc

Задание 1

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

1. Создал новый отдельный файл с тестами функций и методами setup() и teardown() которые создают и удаляют тестовую базу данных:

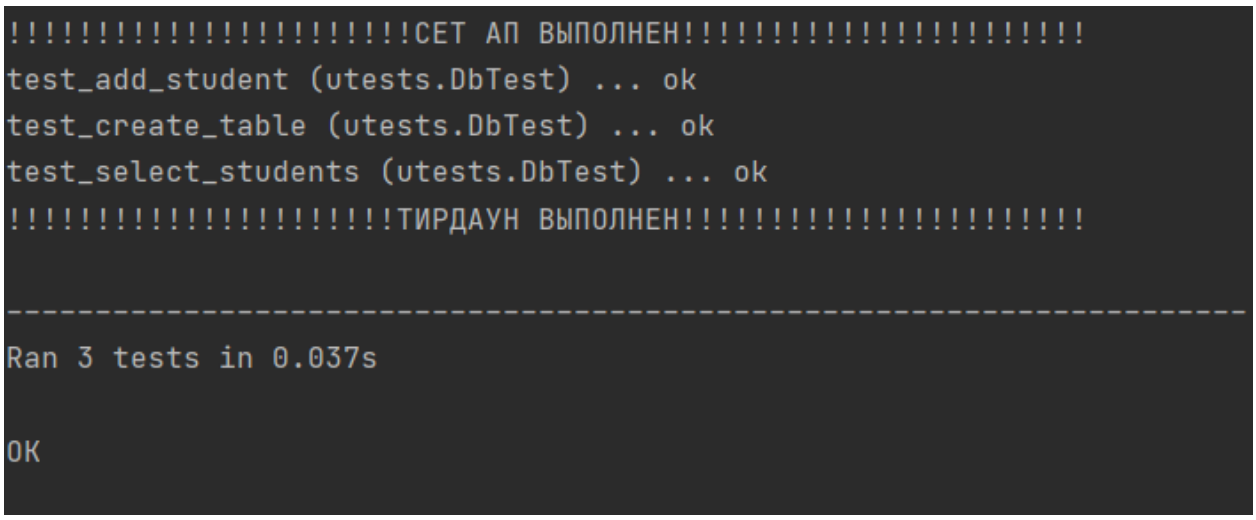


Рисунок 3 – Результат выполнения тестов

Контрольные вопросы:

1. Для чего используется автономное тестирование?

Сущность вашей программы. Автономное тестирование ещё называют модульным или unit-тестированием (unit-testing). Здесь и далее под словом тестирование будет пониматься именно автономное тестирование.

Важной характеристикой unit-теста является его повторяемость, т. е. результат его работы не зависит от окружения (внешнего мира), если же приходится обращаться к внешнем миру в процессе выполнения теста, то необходимо предусмотреть возможность подмены “мира” какой- то статичной сущностью.

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

– unittest

– nose

– pytest

3. Какие существуют основные структурные единицы модуля unittest?

Test fixture

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

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

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

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?

Он представляет собой класс, который должен являться базовым для всех остальных классов, методы которых будут тестировать те или иные автономные единицы исходной программы.

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?

TestCase класс предоставляет набор assert-методов для проверки и генерации ошибок: assertEqual(a, b), assertNotEqual(a, b), assertTrue(x), assertFalse(x), assertIs(a, b), assertIsNot(a, b), assertIsNone(x), assertIsNotNone(x), assertIn(a, b), assertNotIn(a, b), assertIsInstance(a, b), assertNotIsInstance(a, b).

Assert’ы для контроля выбрасываемых исключений и warning’ов: assertRaises(exc, fun, *args, **kwds), assertRaisesRegex(exc, r, fun, *args, **kwds), assertWarns(warn, fun, *args, **kwds), assertWarnsRegex(warn, r, fun, *args, **kwds).

Assert’ы для проверки различных ситуаций: assertAlmostEqual(a, b), assertNotAlmostEqual(a, b), assertGreater(a, b), assertGreaterEqual(a, b), assertLess(a, b), assertLessEqual(a, b), assertRegex(s, r), assertNotRegex(s, r), assertCountEqual(a, b)

Типо-зависимые assert’ы, которые используются при вызове assertEqual(). Приводятся на тот случай, если необходимо использовать

конкретный метод: assertMultiLineEqual(a, b), assertSequenceEqual(a, b), assertListEqual(a, b), assertTupleEqual(a, b), assertSetEqual(a, b), assertDictEqual(a, b).

Дополнительно хотелось бы отметить метод fail(): fail(msg=None). Этот метод сигнализирует о том, что произошла ошибка в тесте.

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. Разберем более подробно методы класса TestSuite.

addTest(test)

Добавляет TestCase или TestSuite в группу.

addTests(tests)

Добавляет все TestCase и TestSuite объекты в группу, итеративно проходя по элементам переменной tests.

run(result)

Запускает тесты из данной группы.

countTestCases()

Возвращает количество тестов в данной группе (включает в себя как отдельные тесты, так и подгруппы).

10. Каково назначение класса TestResult?

Класс TestResult используется для сбора информации о результатах прохождения тестов

11. Для чего может понадобиться пропуск отдельных тестов?

unittest предоставляет нам инструменты для удобного управление процессом пропуска тестов. Это может быть ещё полезно в том плане, что информацию о пропущенных тестах (их количестве) можно дополнительно получить через специальный API, предоставляемый классом TestResult

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

Для безусловного пропуска тестов применяется декоратор: @unittest.skip(reason)

Для условного пропуска тестов применяются следующие декораторы: @unittest.skipIf(condition, reason)

Тест будет пропущен, если условие (condition) истинно:

@unittest.skipUnless(condition, reason)

Условный пропуск тестов можно использовать в ситуациях, когда те или иные тесты зависят от версии программы, например: в новой версии уже не поддерживается часть методов; или тесты могут быть платформозависимые, например: ряд тестов могут выполняться только под операционной системой MS Windows. Условие записывается в параметр condition, текстовое описание – в reason.

Для пропуска классов используется декоратор: @unittest.skip(reason), который записывается перед объявлением класса. В результате все тесты из данного класса не будут выполнены.

Вывод: в ходе работы приобрёл навыки по работе с тестированием при написании программ с помощью языка программирования Python версии 3.