GeekBrains

Основы тестирования

Урок 14

Содержание урока

План курса

1. Основы Python
2. Простые типы данных
3. Коллекции
4. Функции
5. Итераторы и генераторы
6. Модули
7. Файлы и файловая система
8. Сериализация
9. Декораторы
10. ООП. Начало
11. ООП. Особенности Python
12. ООП. Финал
13. Исключения
14. Основы тестирования
15. Обзор стандартной библиотеки Python

Что будет на уроке сегодня

- 📌 Разберёмся с написанием тестов в Python
- 📌 Изучим возможности doctest
- 📌 Узнаем о пакете для тестирования unittest
- 📌 Разберёмся с тестированием через pytest

Основы тестирования

Основы тестирования

- **Функциональное тестирование**
  - Модульное (компонентное)
  - Интеграционное
  - Системное
  - Регрессионное
  - Приемочное
  - Смоук
- **Тестирование производительности**
  - Тестирование отказоустойчивости
  - Нагрузочное
  - Объемное
  - Тестирование масштабируемости
- **Обслуживание (регресс и обслуживание)**
  - Регрессионное
  - Тестирование технического обслуживания

Основы doctest

Основы doctest

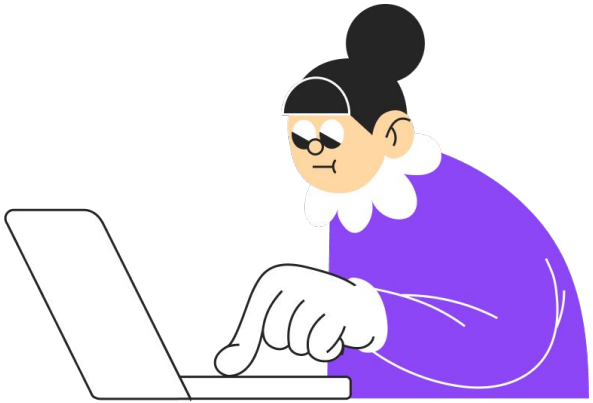
doctest подходит для следующих задач

- Проверка актуальности строк документации модуля путем проверки того, что все интерактивные примеры по-прежнему работают в соответствии с документацией.
- Для регрессионного тестирования. Чтобы убедиться, что интерактивные примеры из тестового файла или тестового объекта работают должным образом.
- Позволяют написать учебную документацию для пакета, обильно иллюстрированную примерами ввода-вывода. В зависимости от того, что выделено — примеры или пояснительный текст, это может быть что-то вроде “грамотного тестирования” или “исполняемой документации”.

Разработка через тестирование, TDD

В TDD (англ. test-driven development) выделяют следующие этапы:

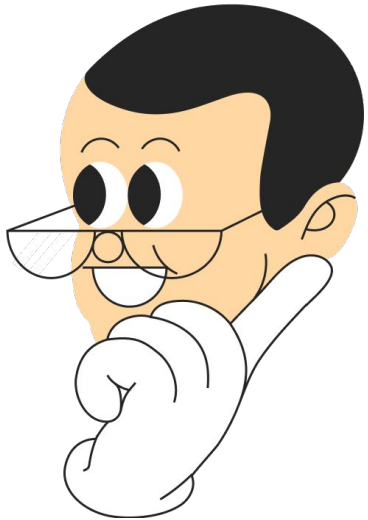
- Добавление теста
- Запуск всех тестов: убедиться, что новые тесты не проходят
- Написать код
- Запуск всех тестов: убедиться, что все тесты проходят
- Рефакторинг
- Повторить цикл



Проверка исполняемой документации

testfile запускает тесты внутри файлов документации

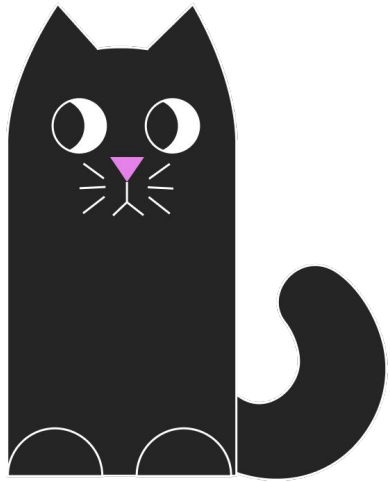
```
import doctest

doctest.testfile('doc_file.txt', verbose=True)
```



Запуск тестов из командной строки

Для запуска тестов зачастую используют вызов из терминала ОС

- `$ python3 -m doctest .\prime.py`
- `$ python3 -m doctest .\prime.py -v`
- `$ python3 -m doctest .\prime.md`
- `$ python3 -m doctest .\prime.md -v`

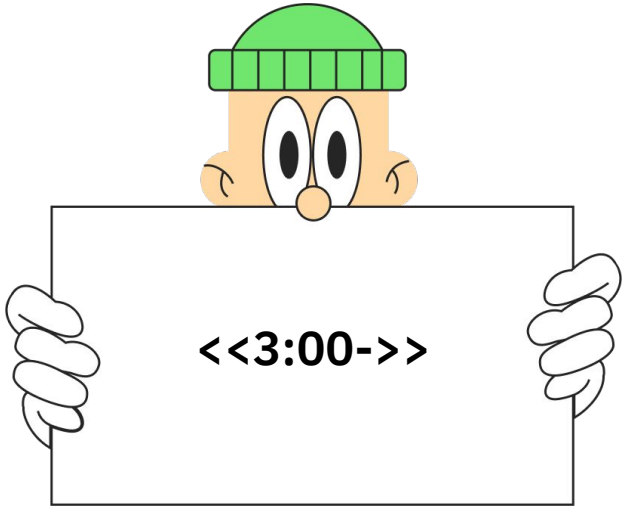


Перед вами несколько строк кода.
Напишите в чат что должна делать программа, чтобы пройти тесты.

У вас 3 минуты.

Основы doctest

```
"""
>>> say('Hello')
Hello Hello
>>> say('Hi', 5)
Hi Hi Hi Hi Hi
>>> say('cat', 3, '(=^.^=)'
cat(=^.^=)cat(=^.^=)cat
"""
```



Основы тестирования с unittest

Общие моменты работы с unittest

```
import unittest

class TestCaseName(unittest.TestCase):

    def test_method(self):
        self.assertEqual(2 * 2, 5, msg='Видимо и в этой
вселенной не работает :-(')

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

```
$ python3 -m unittest tests.py -v
```

Сравнение тестов doctest и unittest

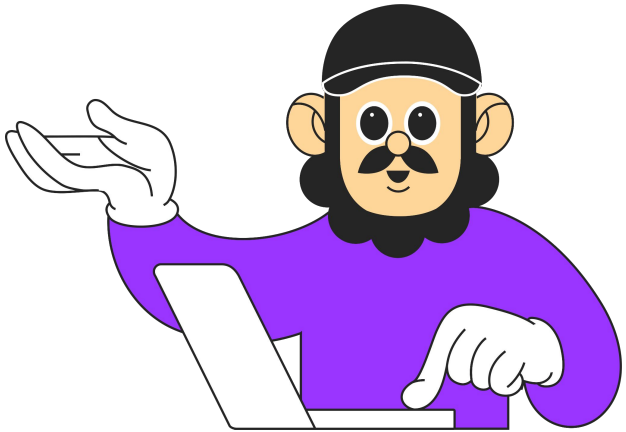
Чтобы легче разобраться в работе unittest сравним его с уже знакомым doctest.

Кстати, тесты doctest можно запускать через unittest

Подготовка теста и сворачивание работ

💡 **def setUp(self) -> None:**
метод setUp выполняется перед запуском каждого теста

💡 **def tearDown(self) -> None:**
метод tearDown выполняется, если был выполнен setUp
независимо от того пройден тест или провален



Перечень доступных утверждений assert

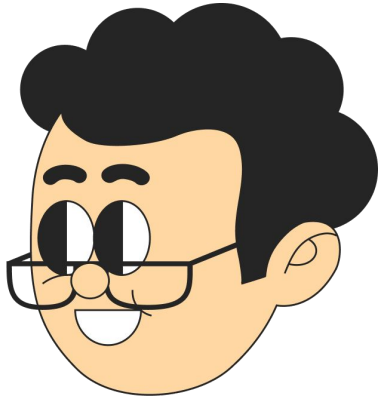
- assertEqual(a, b) - a == b
- assertNotEqual(a, b) - a != b
- assertTrue(x) - bool(x) is True
- assertFalse(x) - bool(x) is False
- assertIs(a, b) - a is b
- assertIsNot(a, b) - a is not b
- assertIsNone(x) - x is None
- assertIsNotNone(x) - x is not None
- assertIn(a, b) - a in b
- assertNotIn(a, b) - a not in b
- assertIsInstance(a, b) - isinstance(a, b)
- assertNotIsInstance(a, b) - not isinstance(a, b)

Перечень доступных утверждений assert

- assertRaises(exc, fun, *args, **kwds) — функция fun(*args, **kwds) поднимает исключение exc
- assertRaisesRegex(exc, r, fun, *args, **kwds) — функция fun(*args, **kwds) поднимает исключение exc и сообщение совпадает с регулярным выражением r
- assertWarns(warn, fun, *args, **kwds) — функция fun(*args, **kwds) поднимает предупреждение warn
- assertWarnsRegex(warn, r, fun, *args, **kwds) — функция fun(*args, **kwds) поднимает предупреждение warn и сообщение совпадает с регулярным выражением r
- assertLogs(logger, level) — блок with записывает логи в logger с уровнем level
- assertNoLogs(logger, level) — блок with не записывает логи в logger с уровнем level

Перечень доступных утверждений assert

- assertAlmostEqual(a, b) - round(a-b, 7) == 0
- assertNotAlmostEqual(a, b) - round(a-b, 7) != 0
- assertGreater(a, b) - a > b
- assertGreaterEqual(a, b) - a >= b
- assertLess(a, b) - a < b
- assertLessEqual(a, b) - a <= b
- assertRegex(s, r) - r.search(s)
- assertNotRegex(s, r) - not r.search(s)
- assertCountEqual(a, b) - a и b содержат одни и те же элементы в одинаковом количестве независимо от их порядка в коллекциях

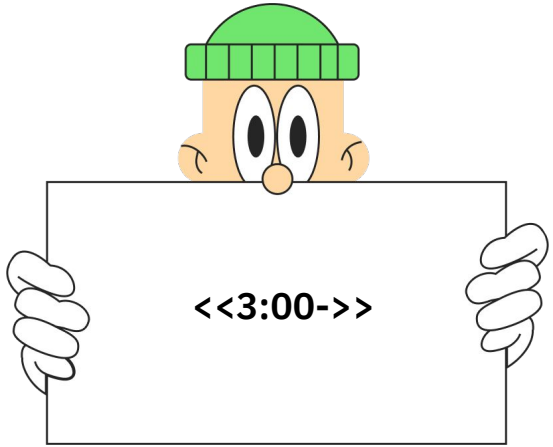


Перед вами несколько строк кода.
Напишите в чат что должна делать программа, чтобы пройти тесты.

У вас 3 минуты.

Основы тестирования с unittest

```
import unittest
from main import func

class TestSample(unittest.TestCase):

    def setUp(self) -> None:
        self.data = { 'one': 1, 'two': 2, 'three': 3, 'four': 4}

    def test_step_1(self):
        self.assertEqual(func(self.data),  4)

    def test_step_2(self):
        self.assertEqual(func(self.data, first= False), 2)

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



Основы тестирования с pytest

Основы тестирования с pytest

```
pip install pytest
```

Код с assert:

```
assert утверждение, "Ложное утверждение"
```

Код с if вместо assert:

```
if утверждение:
    pass
else:
    raise AssertionError("Ложное утверждение")
```

```
$ python3 -m pytest tests.py -vv
$ pytest tests_pt.py
```

Сравнение тестов pytest с doctest и unittest

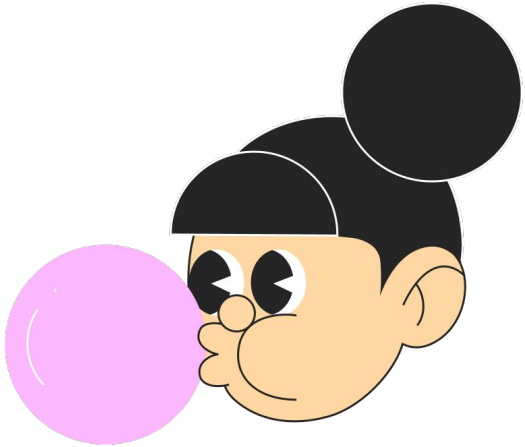
Чтобы легче разобраться в работе pytest сравним его с уже знакомым doctest и unittest.

Кстати, тесты doctest и unittest можно запускать через pytest

pytest fixture

Фикстуры pytest не только заменяют setUp и tearDown, но и дают более широкие возможности тестирования

```
import pytest


@pytest.fixture
def fix():
    ...
    return result


def test_append(fix):
    # обращаемся к result из fix по имени “fix”
    ...
```

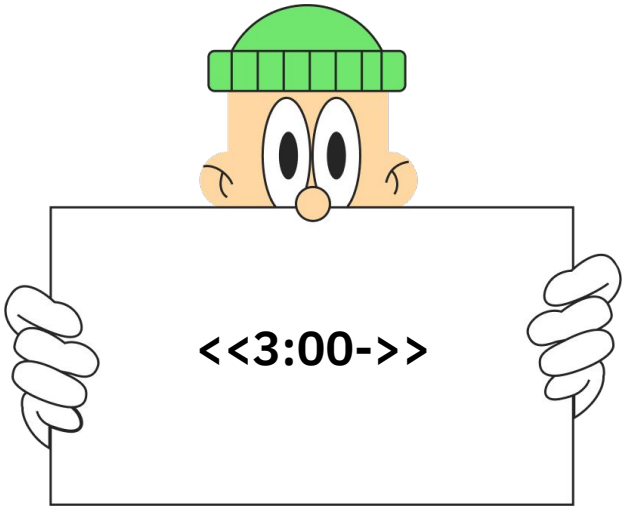


Перед вами несколько строк кода.
Напишите в чат что должна делать программа, чтобы пройти тесты.

У вас 3 минуты.

Основы тестирования с pytest

```
import pytest
from main import func

def test_1():
    assert func(4) == 0

def test_2():
    assert func(4, -4) == (1, 0)

def test_3():
    assert func(4, -10, -50) == (5, -2.5)

def test_4():
    assert func(1, 1, 1) is None

if __name__ == '__main__':
    pytest.main(['-v'])
```



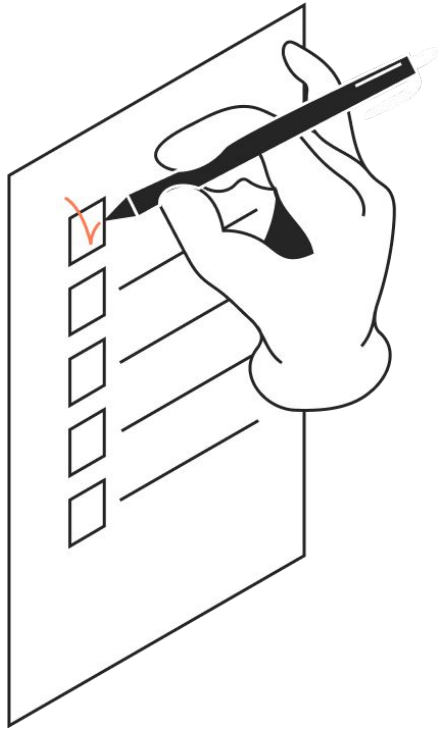
Итоги занятия

На этой лекции мы

- Разобрались с написанием тестов в Python
- Изучили возможности doctest
- Узнали о пакете для тестирования unittest
- Разобрались с тестированием через pytest

Задание

Возьмите 1-3 задачи из прошлых занятий. Напишите к ним тесты. Попробуйте написать одинаковые тесты в трёх инструментах. Так у вас будет возможность сравнить их.



Спасибо за внимание