Advance Python

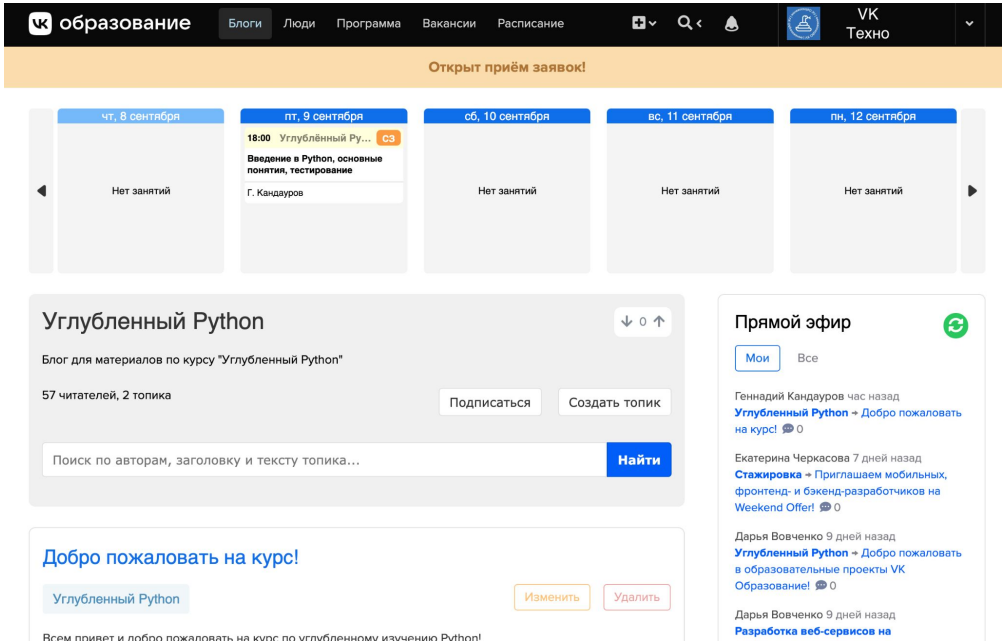
Лекция 7
Тестирование

Кандауров Геннадий

образование

Напоминание отметиться на портале

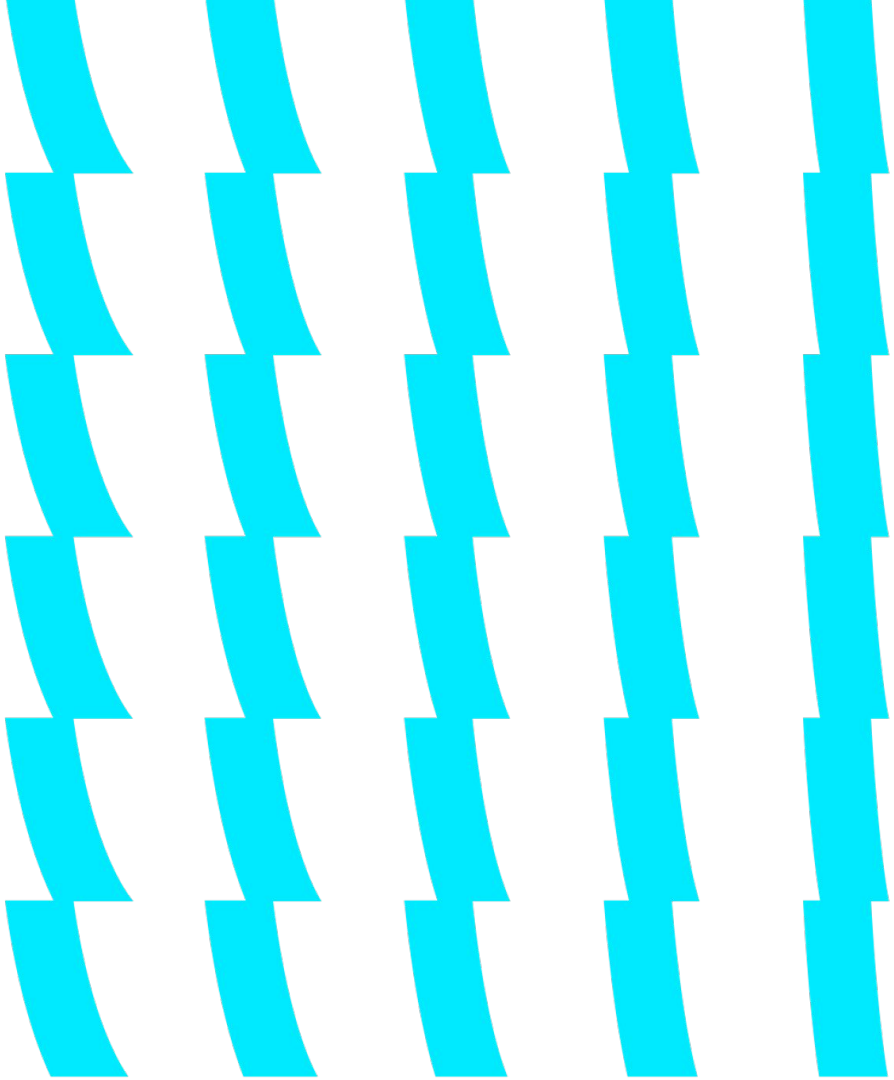
+ оставить отзыв



Содержание занятия

1. Тестирование
2. unittest
3. pytest

Тестирование



Тестирование показывает присутствие ошибок, а не их отсутствие.

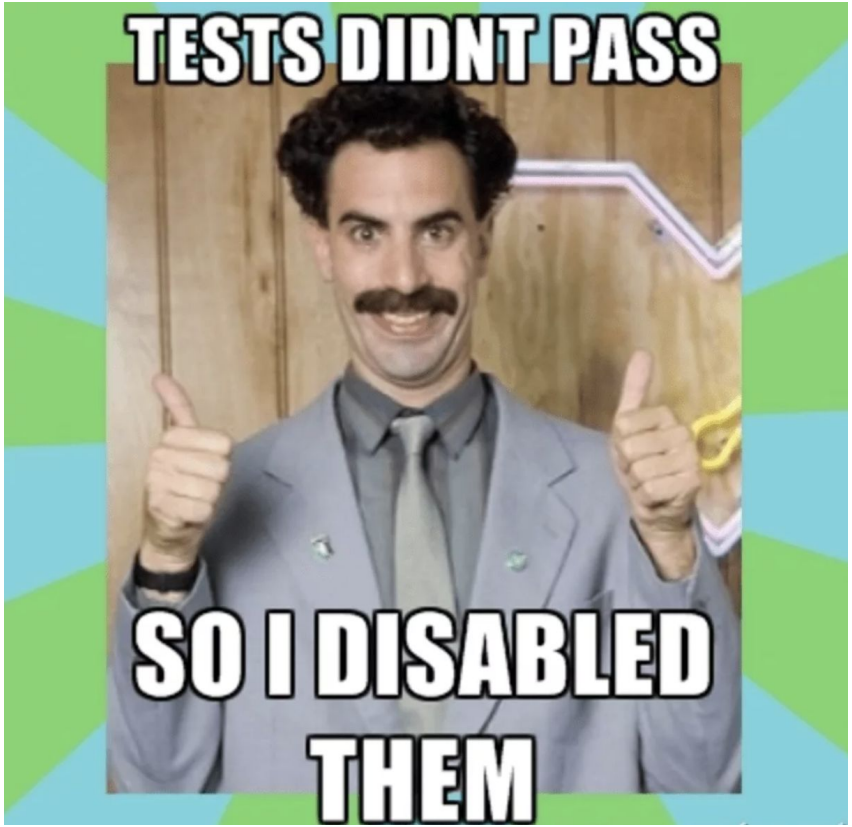
Эдсгер Дейстра

Тестированием можно доказать неправильность программы, но нельзя доказать её правильность.

Цели тестирования

- Проверка правильности реализации
- Проверка граничных условий
- Проверка обработки внештатных ситуаций
- Подготовка ко внесению изменений
- Минимизация последствий

Цели тестирования



Виды тестирования

- Unit-тесты (модульные тесты)
- Функциональное тестирование
- Системное тестирование
- Интеграционное тестирование
- Регрессионное тестирование
- Тестирование производительности
  - Нагрузочное
  - Стресс
- White box, black box

TDD

TDD (Test Driven Development) – техника разработки ПО, основывается на повторении коротких циклов разработки: пишется тест, покрывающий желаемое изменение, затем пишется код, который позволит пройти тест, и далее проводится рефакторинг нового кода.

TDD: алгоритм

1. Пишется тест на функцию/класс
2. Проверяется, что тесты упали (кода еще нет)
3. Пишется код функции/класса для прохождения тестов
4. Проверяется прохождение тестов
5. На этом шаге можно задуматься о качестве кода, провести рефакторинг
6. Прогоняются тесты

Степень покрытия тестами

`coverage` - библиотека для проверки покрытия тестами.

```
pip install coverage
```

```
coverage run tests.py
coverage report -m
coverage html
```

doctest

```
def multiply(a, b):
    """
    >>> multiply(4, 3)
    12
    >>> multiply("a", 3)
    'aaa'
    """
    return a * b
```

```
python -m doctest <file>
```

unittest

`class TestCase`

- `def setUp(self):`

  установки запускаются перед каждым тестом
- `def tearDown(self):`

  очистка после каждого метода
- `def test_<название теста>(self):`

  `код теста`

unittest: TestCase

```
import unittest

class TestString(unittest.TestCase):

    def test_upper(self):

        self.assertEqual("text".upper(), "TEXT")


if __name__ == "__main__":

    unittest.main()
```

unittest: запуск тестов

```
# Найти и выполнить все тесты

python -m unittest discover

# Тесты нескольких модулей

python -m unittest test_module1 test_module2

# Тестирование одного кейса - набора тестов

python -m unittest tests.SomeTestCase

# Тестирование одного метода

python -m unittest tests.SomeTestCase.test_some_method
```

unittest: набор assert*

- `assertEqual(a, b)`
- `assertNotEqual(a, b)`
- `assertTrue(x)`
- `assertFalse(x)`
- `assertIsNone(x)`
- `assertIs(a, b)`
- `assertIsNot(a, b)`
- `assertIn(a, b)`
- `assertIsInstance(a, b)`
- `assertLessEqual(a, b)`
- `assertListEqual(a, b)`
- `assertDictEqual(a, b)`
- `assertRaises(exc, fun, *args, **kwargs)`

unittest: mock

Mock – это объект-пустышка, который заменяет некий реальный объект (функцию, класс, экземпляр, атрибут) для определенной части программы.

- Высокая скорость
- Избежание нежелательных побочных эффектов во время тестирования
- Позволяет задать специальное поведение в рамках теста

```
from unittest.mock import patch

class TestUserSubscription(TestCase):
    @patch("users.views.get_status", return_value=True)
    def test_subscription(self, get_status_mock):
        ...
```

unittest: mock

```
# account_lib.py
import fetch_salary
def calc_income(name, bonus):
    resp = fetch_salary(name)  # request to some API
    return resp + bonus


class TestIncome(unittest.TestCase):
    def test_calc_income(self):
        with mock.patch("account_lib.fetch_salary") as m_fetch_sal:
            m_fetch_sal.return_value = 42
            res = calc_income("username", 100)
            self.assertEqual(res, 142)
        self.assertEqual(m_fetch_sal.call_count, 1)
```

unittest: mock

Атрибуты объекта `Mock` с информацией о вызовах

- `called` — вызывался ли объект вообще
- `call_count` — количество вызовов
- `call_args` — аргументы последнего вызова
- `call_args_list` — список всех аргументов
- `method_calls` — аргументы обращений к вложенным методам и атрибутам
- `mock_calls` — то же самое, но в целом и для самого объекта, и для вложенных

```
self.assertEqual(m_fetch_sal.mock_calls, [mock.call("username")])
```

pytest

https://docs.pytest.org/en/7.1.x/

```python
# content of test_sample.py
def inc(x):
    return x + 1

def test_answer():
    assert inc(3) == 5
```

```
$ pytest
test_sample.py:6: AssertionError
FAILED test_sample.py::test_answer - assert 4 == 5

$ pytest -pdb
```

Общие положения

- Разделение на кейсы по функциям
- Говорящее именование тестовых функций
- Префикс `test_` для модулей и пакетов с тестами
- Префикс `test_` для функций или методов тестирования

Плохо:

```
# lru_testing.py
def test_1():
    assert LRU().get(1) is None
    ...
    lru = LRU(50)
    assert lru.get(1) is None
```

Нормально:

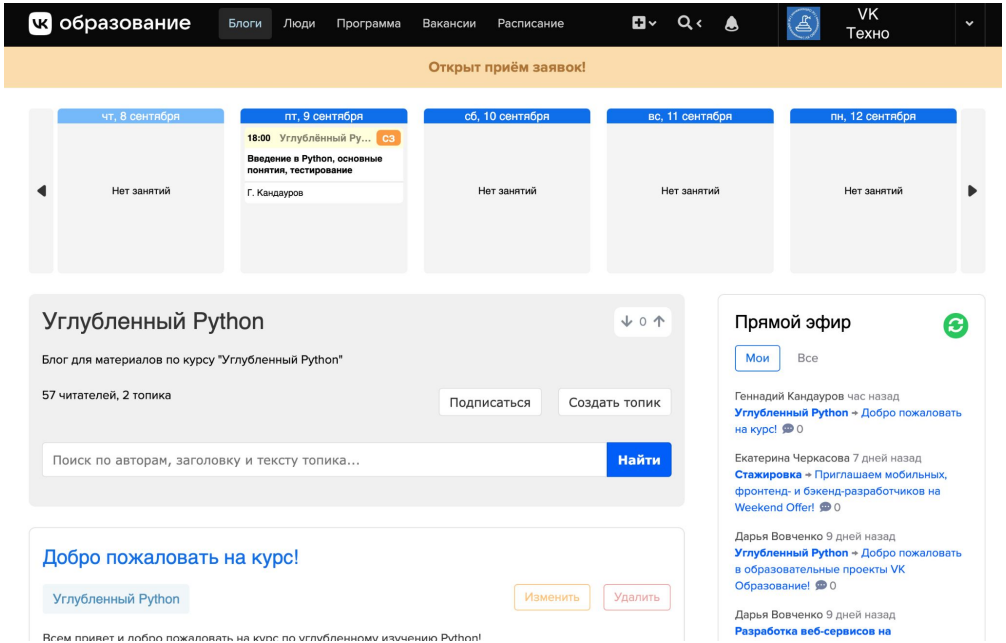
```
# test_lru.py
def test_initial_get():
    assert LRU().get(1) is None
def test_get_big_capacity()
    lru = LRU(50)
    assert lru.get(1) is None
```

Домашнее задание #07

- Функция выставления оценки
- Тесты `int` и `string.partition`
- Проверить и поправить код `flake8` и `pylint`

Напоминание отметиться на портале Vol 2

+ оставить отзыв после лекции



Спасибо за внимание

образование