Интенсив Python

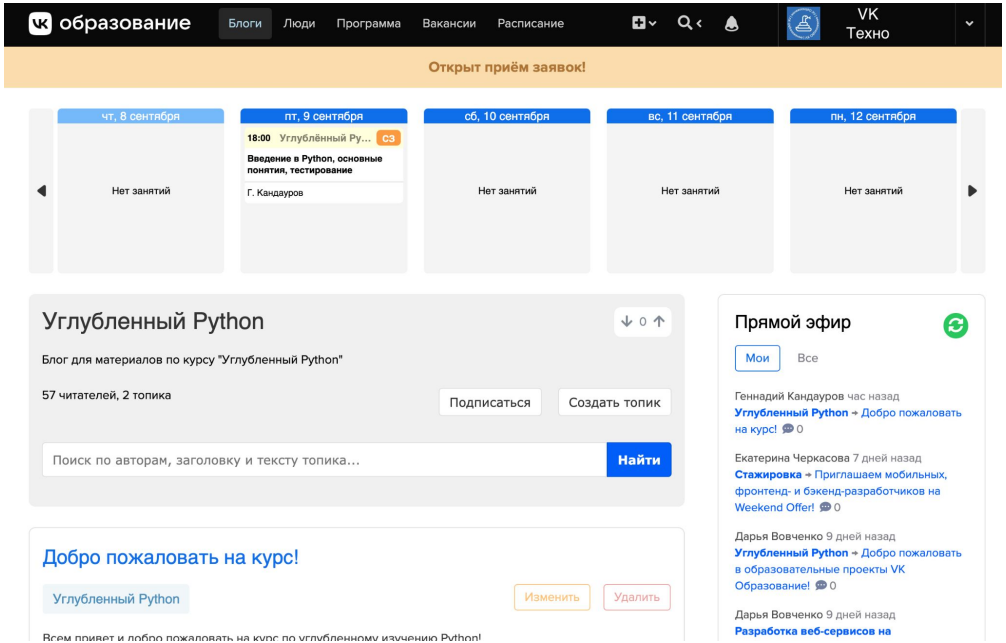
Лекция 5
Тестирование

Кандауров Геннадий

образование

Напоминание отметиться на портале

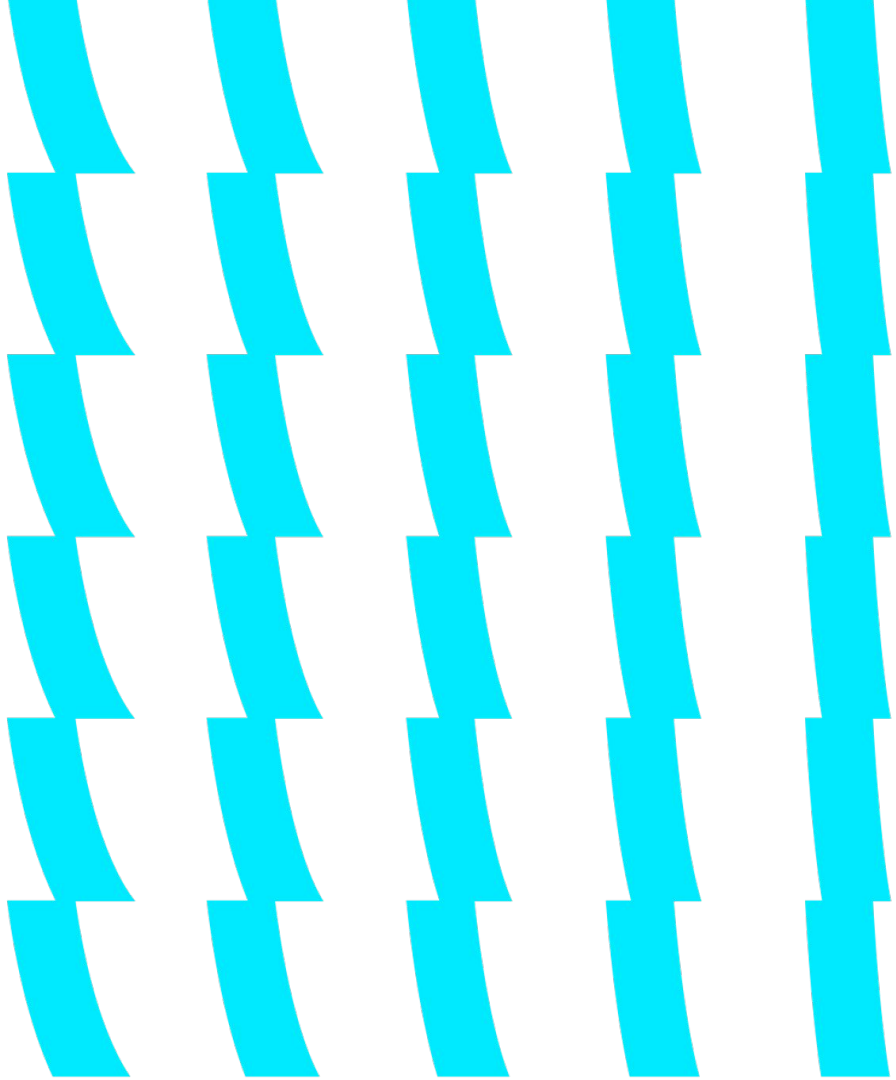
+ оставить отзыв



Содержание занятия

1. Контекстный менеджер
2. Исключения
3. Файлы
4. Форматирование
5. Тестирование

Управляющие конструкции



Управляющие конструкции: контекстный менеджер

```
class CtxManager:
    def __init__(self, name):
        self.db = connect_db(name)
    def __enter__(self):
        return self.db
    def __exit__(self, exc_type, exc_val, exc_tb):
        self.db.close()

# from contextlib import contextmanager

with CtxManager("db_name") as db:
    do_action(db)
```

Управляющие конструкции: исключения

```
try:
    something_dangerous()
except DangerError as err:
    process_danger_error(err)
except TrivialError as err:
    process_trivial_error(err)
except Exception as err:
    process_total_error(err)
else:
    process_no_error()
finally:
    will_be_executed_in_any_case()
```

Управляющие конструкции: исключения

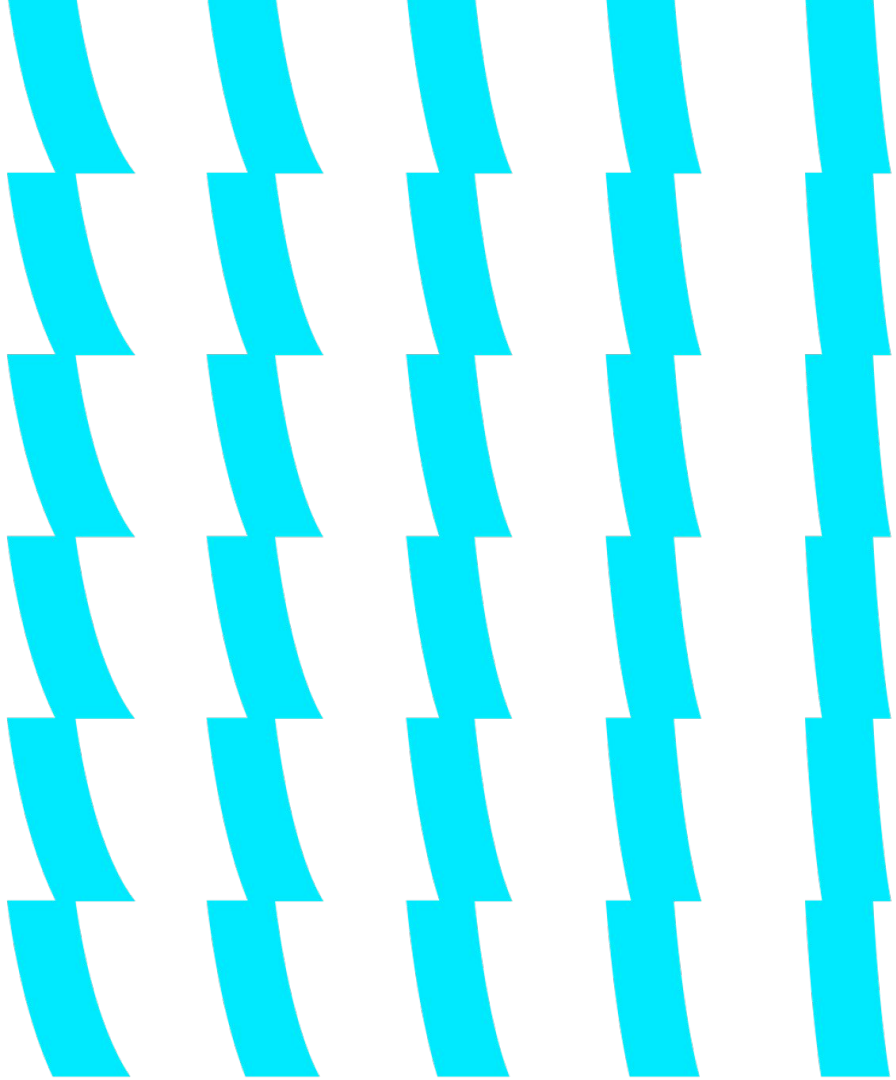
```
# правильно наследоваться от Exception
class TrivialError(Exception): pass

class DangerError(TrivialError): pass

# плохо, только когда реально надо
class UserError(BaseException): pass

# совсем плохо
try:
    pass
except:
    process_err()
```

Работа с файлами



Типы операций с файлами

- связанные с его открытием: открытие, закрытие файла, запись, чтение, перемещение по файлу и др.
- выполняющиеся без его открытия: работа с файлом как элементом файловой системы - переименование, копирование, получение атрибутов и др.

Файловый объект

При открытии файла операционная система возвращает специальный дескриптор файла (идентификатор), однозначно определяющий, с каким файлом далее будут выполняться операции.
В Python работа с файлами осуществляется через специальный абстрактный файловый объект. В зависимости от способа создания такого объекта, он может быть привязан как к физическому файлу на диске, так и другому устройству, поддерживающему схожие операции (стандартный ввод/вывод и пр.).

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,
newline=None, closefd=True, opener=None)

# кодировка
import locale
locale.getpreferredencoding(False)
```

Файловый объект

```
f = open("some.file", "r")
data = f.read()
f.close()

# лучше
with open("some.file", "r") as f:
    data = f.read()
```

Обработка файла

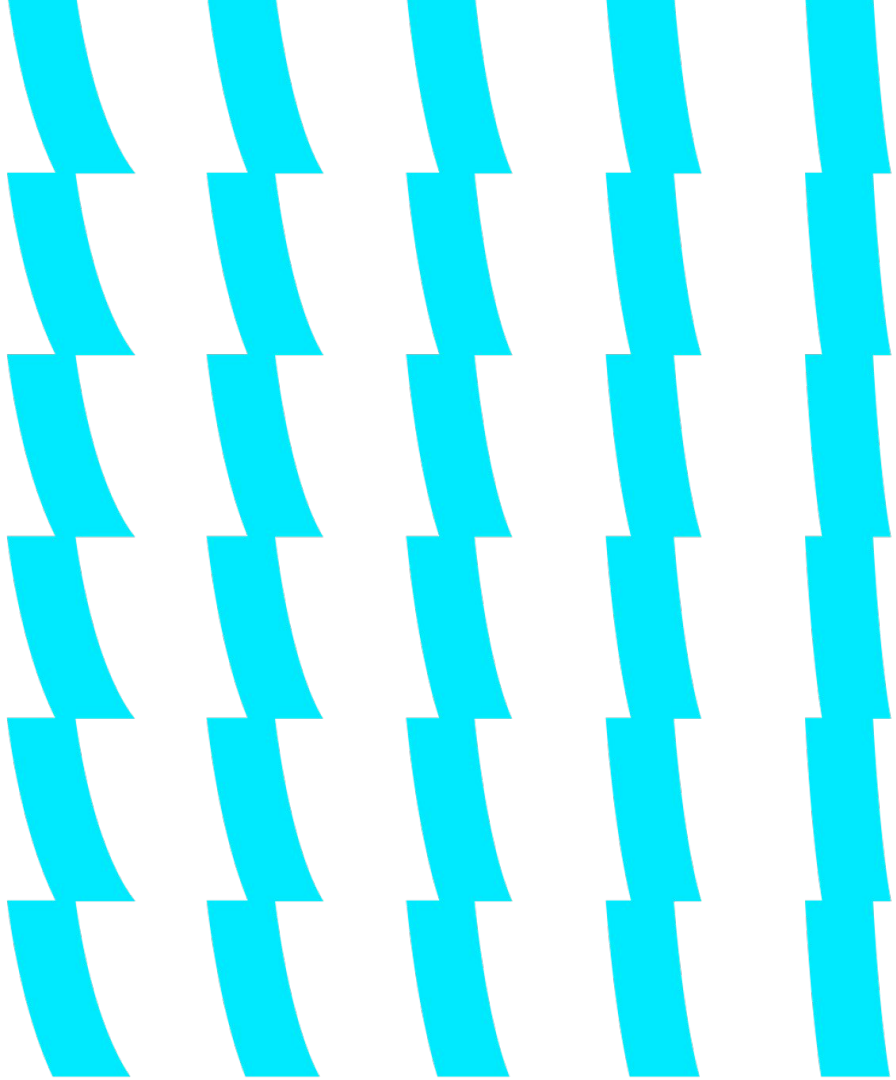
Режим	Описание
r	Только для чтения.
w	Только для записи. Создаст новый файл, если не найдет с указанным именем.
rb	Только для чтения (бинарный).
wb	Только для записи (бинарный). Создаст новый файл, если не найдет с указанным именем.
r+	Для чтения и записи.
rb+	Для чтения и записи (бинарный).
w+	Для чтения и записи. Создаст новый файл для записи, если не найдет с указанным именем.
wb+	Для чтения и записи (бинарный). Создаст новый файл для записи, если не найдет с указанным именем.
a	Откроет для добавления нового содержимого. Создаст новый файл для записи, если не найдет с указанным именем.
a+	Откроет для добавления нового содержимого. Создаст новый файл для чтения записи, если не найдет с указанным именем.
ab	Откроет для добавления нового содержимого (бинарный). Создаст новый файл для записи, если не найдет с указанным именем.
ab+	Откроет для добавления нового содержимого (бинарный). Создаст новый файл для чтения записи, если не найдет с указанным именем.

Файловый объект: методы

- `file.close()`
- `file.fileno()`
- `file.flush()`
- `file.isatty()`
- `file.next()`
- `file.read(n)`
- `file.readline()`
- `file.readlines()`
- `file.seekable()`
- `file.tell()`
- `file.truncate(n)`
- `file.write(str)`
- `file.writelines(sequence)`

Форматирование

https://peps.python.org/pep-0008/



Линтеры

PEP8

Flake8 https://flake8.pycqa.org/en/latest/

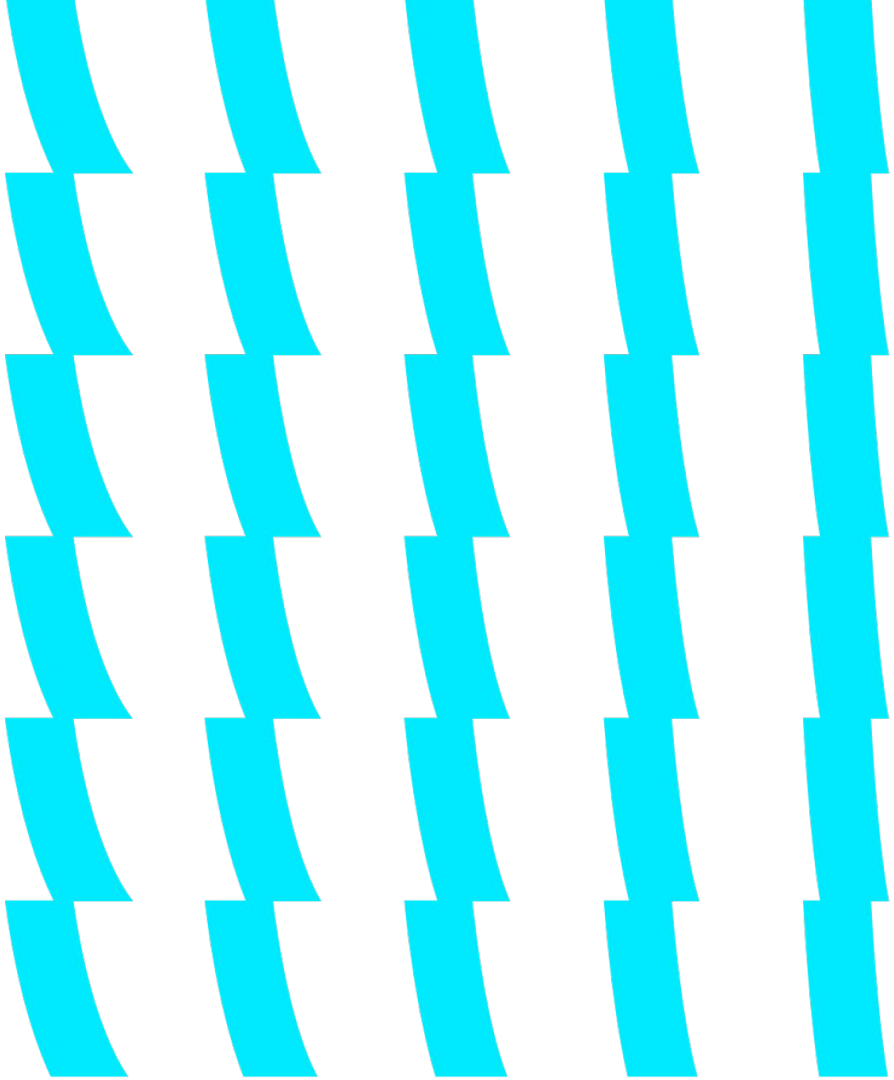
```
$ flake8 .
```

Pylint https://pylint.pycqa.org/en/latest/

```
$ pylint *.py
$ pylint --disable C0114,C0116 *.py
```

black, isort

Тестирование



Тестирование показывает присутствие ошибок, а не их отсутствие.

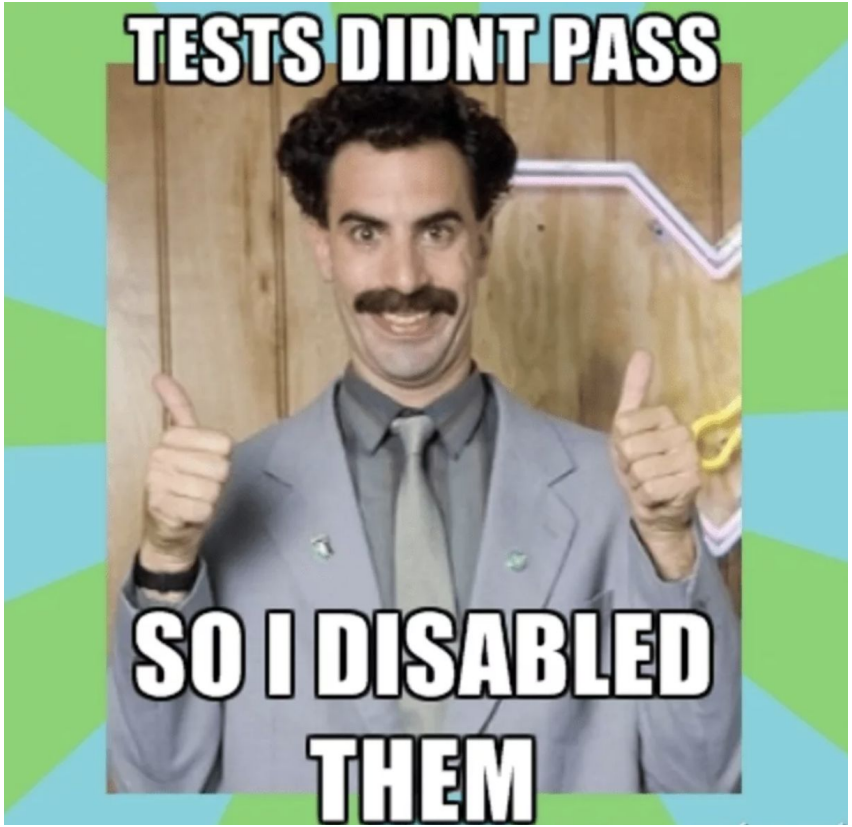
Эдсгер Дейстра

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

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
```

Инструменты тестирования в Python

- doctest
- unittest
- pytest
- factory_boy
- selenium

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/

```
# 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
```

factory_boy

Библиотека `factory_boy` служит для генерации разнообразных объектов (в т.ч. связанных) по заданным параметрам.

https://factoryboy.readthedocs.io/en/stable/

https://faker.readthedocs.io/en/master/

```
pip install factory_boy
```

selenium

Selenium `WebDriver` – это программная библиотека для управления браузерами. WebDriver представляет собой драйверы для различных браузеров и клиентские библиотеки на разных языках программирования, предназначенные для управления этими драйверами.

```
pip install selenium
```

selenium

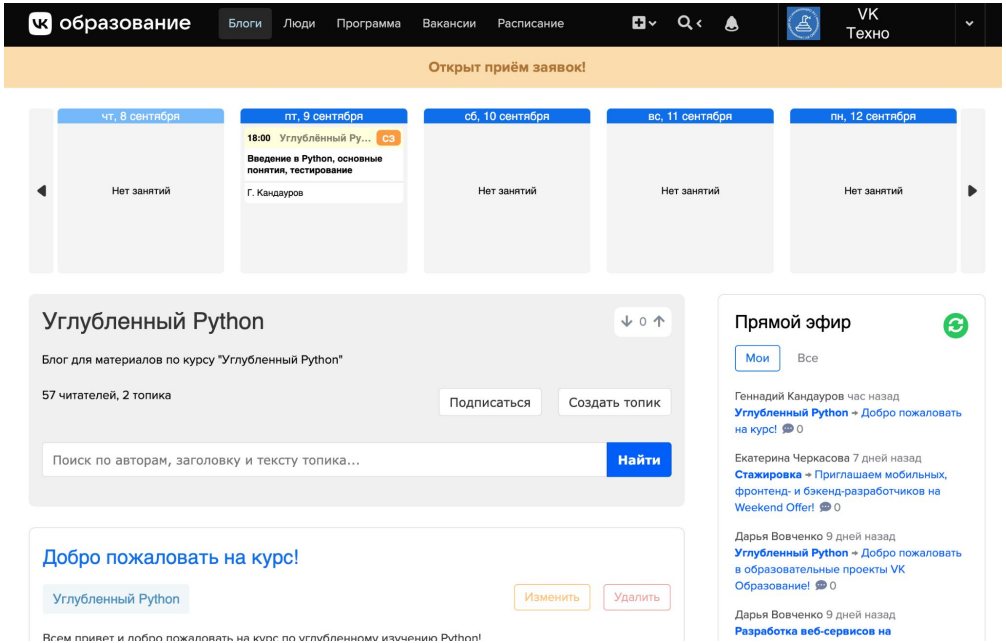
- Требует конкретного драйвера для конкретного браузера (Chrome, Firefox и т.д.)
- Автоматическое управление браузером
- Поддержка Ajax
- Автоматические скриншоты

Домашнее задание #05

- Реализовать игру крестики-нолики
- Написать тесты (`unittest` или `pytest`)
- Проверить покрытие тестов через `coverage`
- Проверить и поправить код `flake8` и `pylint`

Напоминание отметиться на портале Vol 2

+ оставить отзыв после лекции



Спасибо за внимание

образование