

# РАЗРАБОТКА ТЕСТОВ



Адилет  
Асанкожоев



**Адилет Асанкожоев**

Python-разработчик в Makers.kg



# План занятия

1. [Что такое тестирование?](#)
2. [Виды тестирования](#)
3. [Когда и зачем используются тесты](#)
4. [Тест-фреймворки python](#)
5. [Примеры тестов](#)



# Что такое тестирование?

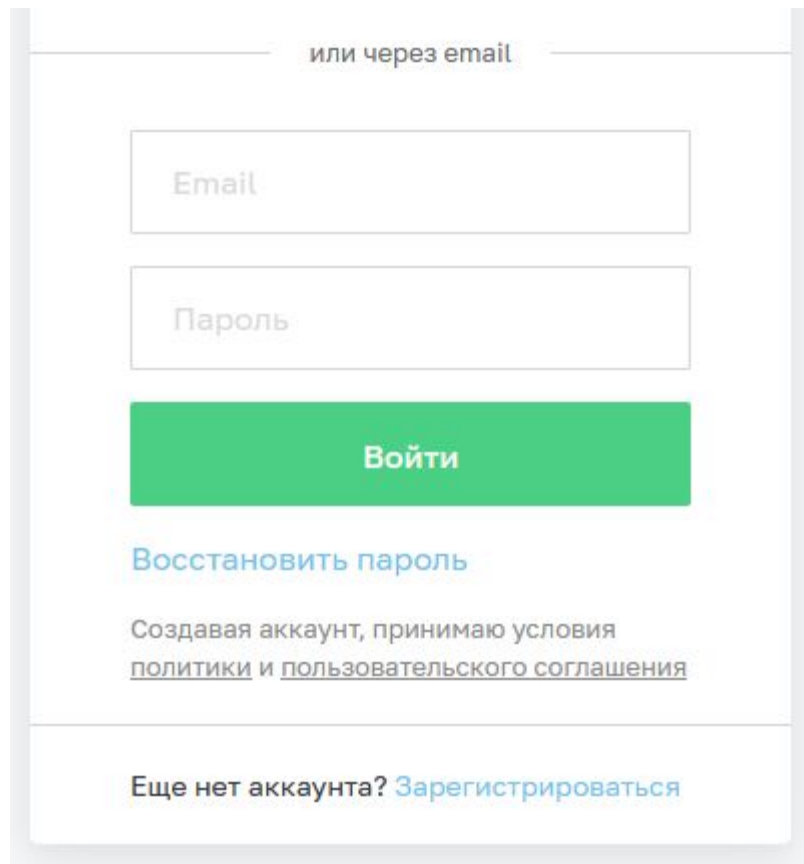


# Что такое тестирование?

Тестирование ПО — процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определённым образом (ISO/IEC TR 19759:2005).

# Тестирование

Какие тесты вы применили бы к этой форме?



или через email

Email

Пароль

Войти

[Восстановить пароль](#)

Создавая аккаунт, принимаю условия [политики](#) и [пользовательского соглашения](#)

Еще нет аккаунта? [Зарегистрироваться](#)

The image shows a login form with a light gray background. At the top, there is a link 'или через email'. Below it are two input fields: 'Email' and 'Пароль'. A green button labeled 'Войти' is positioned below the password field. Underneath the button is a blue link 'Восстановить пароль'. Further down is a line of text: 'Создавая аккаунт, принимаю условия' followed by two underlined blue links: 'политики' and 'пользовательского соглашения'. At the bottom, there is a line of text: 'Еще нет аккаунта?' followed by a blue link 'Зарегистрироваться'.

# Тестирование

Какие тесты вы применили бы к этой форме?

1. логин и пароль
2. логин и пароль со спец. символами
3. логин и пароль от другого логина
4. логин и неправильный пароль
5. пустые поля
6. несуществующий логин

или через email

Email

Пароль

Войти

[Восстановить пароль](#)

Создавая аккаунт, принимаю условия [политики](#) и [пользовательского соглашения](#)

Еще нет аккаунта? [Зарегистрироваться](#)



# Виды тестирования



# Виды тестирования

- **Ручное** (выполнение сценариев использования тестировщиком/пользователем)
- **Нагрузочное** (на стрессоустойчивость, загружая объемы данных или количество запусков/подключений)
- **Интеграционное** (проверка установки, взаимодействия с внешними сервисами и пр.)
- **Регрессия** (не сломались уже существующие тесты, верно сохранилась старая логика)
- **Smoke** (поверхностные быстрые тесты на то, что программа в принципе запускается как надо)
- **Модульное или юнит-тестирование** (проверка конкретного модуля/класса, логическую единицу кода)

Полезные ссылки про виды тестирования: [rosalab](#) и [protesting](#)



## Зачем и почему?

- Автоматическая проверка того, что всё работает именно так, как задумано.
- Избегая негативного влияния (регрессии) новых патчей на боевой код, ловим ошибки перед публикацией или во время разработки.
- При разработке в команде, тесты помогают понять, что ваш новый код не поломал логику ваших коллег.
- Если, читая чужой код, не ясно, как его использовать, тесты помогают разобраться, как он должен работать.



# Test-фреймворки в Python



# Тест-фреймворки python

- pytest
- unittest
- doctest
- nose

Взаимодействие тестов с web-браузером:

- selenium

---

# unittest vs pytest

## unittest:

- Входит в стандартную библиотеку Python
- Выполнен в стиле xUnit
- Удобен для unit-тестирования
- Есть специальные методы для проверок вместо выражения `assert`

## pytest:

- Нужно устанавливать `pip install pytest`
- Проще и мощнее, чем unittest. По духу ближе к python
- Совместим с unittest.
- Любая функция или класс, начинающийся слова `test_` будет тестом.
- Большая экосистема. Сотни плагинов.



## Элементы unittest и pytest

- **test case** — базовый класс для тестов со встроенными методами для подготовки данных и проверок;
- **test suite** — группировка и порядок исполнения тестов для лучшей организованности большого количества;
- **test fixture** — инструмент для загрузки тестовых данных из файла json и пр. форматов;
- **test runner** — вариативность запуска тестов, поиск всех тестов в файле/каталоге.

# Служебные методы unittest.TestCase

Методы:

- **SetUp** — запускается перед выполнением каждого теста в классе
- **TearDown** — запускается после каждого теста в классе
- **SetUpClass** — перед запуском тестового класса
- **TearDownClass** — по завершению всех тестов в классе
- **skipTest(reason)** — вызывается, чтобы пропустить текущий тест функции:
- **SetUpModule** — перед запуском каких-либо тестов в модуле
- **TearDownModule** — по завершению всех тестов в модуле

Декораторы:

- **skipIf / skipUnless** — пропустить тест по условию



# Примеры тестов



# Пример теста с помощью unittest

```
import unittest

def multiplication_int(a, b):
    return a * b

def multiplication_string(line, n):
    return line * n

class TestSomething(unittest.TestCase):
    def setUp(self):
        print("method setUp")

    def tearDown(self):
        print("method tearDown")

    def test_numbers_3_4(self):
        self.assertEqual(multiplication_int(3, 4), 12)

    def test_strings_a_3(self):
        self.assertEqual(multiplication_string('a', 3), 'aaa')

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

# Пример теста с помощью pytest

```
def multiplication_int(a, b):  
    return a * b  
  
def multiplication_string(line, n):  
    return line * n  
  
class TestSomething:  
  
    def setup(self):  
        print("method setup")  
  
    def teardown(self):  
        print("method teardown")  
  
    def test_numbers_3_4(self):  
        assert multiplication_int(3, 4) == 12  
  
    def test_strings_a_3(self):  
        assert multiplication_string('a', 3) == 'aaa'
```

# Проверка значений

Method	Checks that
<code>assertEqual(a, b)</code>	<code>a == b</code>
<code>assertNotEqual(a, b)</code>	<code>a != b</code>
<code>assertTrue(x)</code>	<code>bool(x) is True</code>
<code>assertFalse(x)</code>	<code>bool(x) is False</code>
<code>assertIs(a, b)</code>	<code>a is b</code>
<code>assertIsNot(a, b)</code>	<code>a is not b</code>
<code>assertIsNone(x)</code>	<code>x is None</code>
<code>assertIsNotNone(x)</code>	<code>x is not None</code>
<code>assertIn(a, b)</code>	<code>a in b</code>
<code>assertNotIn(a, b)</code>	<code>a not in b</code>
<code>assertIsInstance(a, b)</code>	<code>isinstance(a, b)</code>
<code>assertNotIsInstance(a, b)</code>	<code>not isinstance(a, b)</code>

\*Методы кликабельны

# Сравнение значений

Method	Checks that
<code>assertAlmostEqual(a, b)</code>	<code>round(a-b, 7) == 0</code>
<code>assertNotAlmostEqual(a, b)</code>	<code>round(a-b, 7) != 0</code>
<code>assertGreater(a, b)</code>	<code>a &gt; b</code>
<code>assertGreaterEqual(a, b)</code>	<code>a &gt;= b</code>
<code>assertLess(a, b)</code>	<code>a &lt; b</code>
<code>assertLessEqual(a, b)</code>	<code>a &lt;= b</code>
<code>assertRegex(s, r)</code>	<code>r.search(s)</code>
<code>assertNotRegex(s, r)</code>	<code>not r.search(s)</code>
<code>assertCountEqual(a, b)</code>	<i>a</i> and <i>b</i> have the same elements in the same number, regardless of their order.

# Проверка объектов

Method	Used to compare
<code>assertMultiLineEqual(a, b)</code>	strings
<code>assertSequenceEqual(a, b)</code>	sequences
<code>assertListEqual(a, b)</code>	lists
<code>assertTupleEqual(a, b)</code>	tuples
<code>assertSetEqual(a, b)</code>	sets or frozensets
<code>assertDictEqual(a, b)</code>	dicts

# Test case decorators

**Пропустить тест по условию:**

`@unittest.skipIf(condition, reason)`

`@unittest.skipUnless(condition, reason)`

**Ожидаем сбой в тесте:**

`@unittest.expectedFailure`

*Если тест не пройден, он будет считаться успешным.*

*Если тест пройден, он будет считаться неудачным.*

**Пропустить тест в любом случае:**

`@unittest.skip(reason)`

*При временной потере актуальности теста, но сохранить как пример.*

# Пример теста

```
import sys
import unittest
import requests

class MyTestCase(unittest.TestCase):

    @unittest.skip("demonstrating skipping")
    def test_nothing(self):
        self.fail("shouldn't happen")

    @unittest.skipIf(requests.__version__ < 3, "unsupported library version")
    def test_format(self):
        # Код теста, который подходит по версии
        pass

    @unittest.skipUnless(sys.platform.startswith("win"), "requires Windows")
    def test_windows_support(self):
        # Код теста, подходящий только для windows
        pass
```



## Полезные ссылки

- [Документация по unittest](#)
- [Набор различных библиотек для тестирования на Python](#)





# Итоги

Сегодня на занятии мы:

- Узнали, что такое тестирование, какие виды тестирования бывают и почему тестирование важно.
- Познакомились с тестовыми фреймворками: unittest и pytest.
- Разобрали их основные отличия.
- Научились писать тесты на свой код.



# Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задаём в чате !
- Задачи можно сдавать по частям.
- Зачёт по домашней работе проставляется после того, как приняты **все задачи**.

**Задавайте вопросы и  
пишите отзыв о лекции!**

**Адилет Асанкожоев**