ozon{tech

PyTest: введение

План занятия

PyTest

PyTest — фреймворк для написания автотестов на Python.

Основные особенности и преимущества фреймворка:

- Автообнаружение тестов вида:
  - файлы test_*.py
  - файлы *_test.py
  - функции test_*
  - классы TestSomething
- Отсутствие boilerplate (шаблонного повторяющегося кода)
- Простая параметризация
- Фикстуры
- Маркеры для группировки тестов

Пример теста:

```
# test_assert.py
def logic():
    return 42


def test_function():
      assert logic() == 41


# E         assert 42 == 41
# E           +42
# E           -41
# test_assert.py: AssertionError
```

Ассерты в тестах нужны, чтобы проверить соответствие ожидаемого результата фактическому. В Python есть встроенный assert (см. выше). В Ozon для ассертов чаще используется библиотека PyHamcrest, ассерты с ней понятнее и легче читаются.

```
assert_that(theBiscuit.isCooked(), 'not cooked')
assert_that(theBiscuit, equal_to(myBiscuit))
assert_that(theBiscuit, is_(equal_to(myBiscuit)))
assert_that(theBiscuit, is_(myBiscuit))
assert_that(theBiscuit, instance_of(Biscuit))
assert_that(theBiscuit, is_(instance_of(Biscuit)))
assert_that(theBiscuit, is_(Biscuit))
```

Маркеры

Маркеры (метки) в PyTest реализованы в виде декораторов. Самые распространенные маркеры:

- @pytest.mark.skip — для пропуска тестов

```
@pytest.mark.skip(reason='empty_test')
def test_skip_example():
    pass
```

- @pytest.mark.xfail — для пометки теста как ожидаемо падающего

```
@pytest.mark.xfail(strict=True,
                   reason='wait for fix',
                   raises=ZeroDivisionError,
                   run=True)
def test_xfail_example():
    assert 0 / 1
```

- @pytest.mark.parametrize — для параметризации тестов

```
@pytest.mark.parametrize('device_id, platform',
                         [(549, 'Android'), (535, 'Ios')],
                         ids=['Android', 'Ios'])
def test_get_mobile_devices(device_id, platform):
    response = ActDeviceApiClient().get_device_by_id(device_id)
    assert_that(response.json()['value']['platform'],
                equal_to(platform))
```

pytest --markers — команда для получения всех меток

Фикстуры

Фикстуры — функции для подготовки тестовых данных и настройки окружения. В PyTest реализованы в виде декоратора @pytest.fixture. В качестве параметра принимают scope, указывающий на уровень фикстуры (время жизни).

Доступные scope:

- function (по умолчанию) — setup перед выполнением тестовой функции, использующей данную фикстуру teardown после выполнения.
- class — setup перед выполнением всех тестов из тестового класса, teardown после выполнения.
- module — setup перед выполнением всех тестов из модуля (файла), teardown после выполнения тестов из модуля.
- package — setup перед выполнением всех тестов из пакета (директории), teardown после выполнения тестов из пакета.
- session — setup перед выполнением первого теста из текущей сессии (запуска), teardown после выполнения всех тестов.

Фикстуры являются расширяемыми: одни фикстуры могут использовать и дополнять другие.

Пример наследования (фикстура device_data использует данные из фикстуры platform и обогащает их):

```
@pytest.fixture()
def platform():
    return random.choice('Linux', 'Android', 'Ios')


@pytest.fixture()
def device_data(platform):
    return {'platform': platform,
            'device_id': get_random_device_id_by_(platform=platform)}


def test_get_device(device_data):
    response = ActDeviceApiClient().get_device_by_id(device_data['device_id'])
    assert_that(response.json()['value']['platform'], equal_to(device_data['platform']))
```

Пример использования scope=’session’. В тесте test_get_device используются фикстуры device_id и platform. При этом все тесты в текущей сессии получат одно значение фикстуры platform и разные значения device_id, так как device_id исполняется перед запуском каждого теста, а platform — только один раз за сессию.

```python
@pytest.fixture(scope='session')
def platform():
    return random.choice('Linux', 'Android', 'Ios')


@pytest.fixture()
def device_id(platform):
    return get_random_device_id_by_(platform=platform)


def test_get_device(device_id, platform):
    response = ActDeviceApiClient().get_device_by_id(device_id)
    assert_that(response.json()['value']['platform'], equal_to(platform))
```

Еще один параметр фикстур — autouse. Он необходим для автоматического запуска фикстур, независимо от того, используется она напрямую или нет. Так, на примере ниже фикстура from_autouse нигде не используется напрямую, но исполняется при запуске тестового класса.

```python
@pytest.fixture(scope='class')
def lst():
    return []


@pytest.fixture(scope='class', autouse=True)
def from_autouse(lst):
    lst.append('from_autouse')


@pytest.fixture(scope='class', autouse=True)
def class_fixture(lst):
    lst.append('class_fixture')


class TestClass:
    def test_lst(self, lst, class_fixture):
        assert_that(lst, contains_inanyorder('from_autouse', 'class_fixture'))
```

Запуск тестов и конфигурация PyTest

Пример запуска тестов из консоли:

```
> pytest
> pytest folder
> pytest folder -k "Test Keyword"
```

Файл pytest.ini — конфигурационный, указывает на то, с какими флагами будут запускаться тесты, какие директории и пр. Пример: https://docs.pytest.org/en/6.2.x/customize.html#pytest-ini

conftest.py — файл, где хранятся фикстуры. Фикстуры из файла доступны во всех тестах пакета, в котором находится conftest.py.

__init__.py — при добавлении файла в определенную директорию мы превращаем pytest-модуль в пакет (package).

Заключение

На этом занятии мы обсудили:

- Как работать с фреймворком PyTest.
- Как использовать маркеры PyTest.
- Как запускать тесты и конфигурировать тестовый проект.

Материалы

- Официальная документация: https://docs.pytest.org/en/7.1.x/
- Библиотека PyHamcrest: https://pyhamcrest.readthedocs.io/en/release-1.8/tutorial/