Wifi

Tinkoff Guest - tinkoff1

Tinkoff Python

Лекция 2

Модули, пакеты. Автотесты

Ахтаров Данил

Pathlib

https://docs.python.org/3/library/pathlib.html

Без Pathlib

```
1 import os
2
3 directory = '/home/user/temp/'
4 filepath = os.path.join(directory, 'data.csv')
5
6 if os.path.exists(filepath):
7     print('exist')
```

Pathlib

```python
import os

directory = '/home/user/temp/'
filepath = os.path.join(directory, 'data.csv')

if os.path.exists(filepath):
    print('exist')
```

```python
from pathlib import Path

directory = Path('/home/user/temp/')
filepath = directory / 'data.csv'

if filepath.exists():
    print('exist')
```

Pathlib заменит вам

- open
- os.mkdir
- os.rmdir
- os.path.join
- os.path.*
- glob

О чем будем говорить?

- Модули и пакеты в python
- Управление зависимостями
- Декораторы, генераторы
- Написание автотестов

Modules

```
$ ls
foo.py
bar.py
```

https://docs.python.org/3/tutorial/modules.html

Типичный модуль

cat.py <

```
1 def hello(name):
2   print(f'Hello from {name}')
3
4 hello('cat.py')
```

```
$ python cat.py

Hello from cat.py
```

Импорт модуля

cat.py
dog.py <

```
1 from cat import hello
2
3 hello('dog.py')
```

```
$ python dog.py

Hello from cat.py
Hello from dog.py
```

*

cat.py
dog.py <

```
1 from cat import *
2
3 hello('dog.py')
```

```
$ python dog.py

Hello from cat.py
Hello from dog.py
```

Код импортируемого модуля выполнится,

но только один раз!

(во время первого импорта)

__name__

__name__ is a built-in variable which evaluates to the name of the current module

- __main__
- cat
- animals.some_package.dog

if __name__ == "__main__": ...

cat.py < dog.py

```
1 def hello(name):
2   print(f'Hello from {name}')
3
4 if __name__ == '__main__':
5   hello('cat.py')
```

```
$ python dog.py

Hello from dog.py
```

Packages

https://docs.python.org/3/tutorial/modules.html#packages

Типичный пакет

```
animals/
  __init__.py
  cat.py
dog.py <
```

```
1 from animals.cat import hello
2
3 hello('dog.py')
```

```
$ python dog.py

Hello from dog.py
```

__init__.py

```
animals/
  __init__.py <
  cat.py
dog.py
```

```python
from .cat import hello

hello('animals.__init__')
```

```
$ python dog.py

Hello from animals.__init__
Hello from dog.py
```

Скрываем структуру пакета в __init__.py

```
animals/
  __init__.py
  cat.py
dog.py <
```

```
1 from animals.cat import hello
2
3 hello('dog.py')
```

```
$ python dog.py

Hello from animals.__init__
Hello dog.py
```

А что, если мы хотим запустить пакет как скрипт?

__main__.py

```
animals/
  __init__.py
  __main__.py <
  cat.py
dog.py
```

```python
from .cat import hello

if __name__ == '__main__':
  hello('animals.__main__')
```

```
$ python -m animals

Hello from animals.__init__
Hello from animals.__main__
```

Типы импортов

from .cat import hello - относительный

from animals.cat import hello - абсолютный

Где python ищет код, когда мы что-то импортируем?

PYTHONPATH

```
$ python -c "import sys; print(sys.path)"
['', ... '.../python3.7/site-packages']
```

https://docs.python.org/3/library/sys.html#sys.path

Кольцевые импорты

```python
# cat.py
from dog import hello_dog

def hello_cat():
    print('hello_cat')

hello_dog()
```

```python
# dog.py
from cat import hello_cat

def hello_dog():
    print('hello_dog')

hello_cat()
```

```
$ python dog.py

ImportError: cannot import name ...
  (most likely due to a circular import) ...
```

Проблема решается

- Перепланировкой модулей
- Переносом импорта внутрь функции/кода, в которой он необходим
- Ещё некоторыми ужасными способами

Управление зависимостями

PIP - package manager

https://pip.pypa.io/en/stable/

pip install <package>

requirements.txt

```
$ cat requirements.txt
pytest==4.1.1
flake8==3.7
pylint==2.2

# установка всех пакетов из requirements.txt
$ pip instal -r requirements.txt
```

Фиксация зависимостей

requirements.txt

```
$ pip freeze > requirements.txt

$ cat requirements.txt
tomicwrites==1.2.1
attrs==18.2.0
more-itertools==5.0.0
pluggy==0.8.1
py==1.7.0
pytest==4.1.1
six==1.12.0

# установка всех пакетов из requirements.txt
$ pip instal -r requirements.txt
```

Зависимости можно разделять

```
# requirements-dev.txt
-r requirements.txt
pytest==4.1.1
flake8==3.7
pylint==2.2
```

Почему ставить все пакеты глобально — плохая идея?

Почему ставить все пакеты глобально плохая идея?

- Конфликты версий
- Невоспроизводимость окружения

Venv

```
$ python -m venv .venv
```

https://docs.python.org/3/library/venv.html

Venv

```
.venv/
  bin/
    ...
    activate
    python
    python3.8
  lib/
    python3.8/site-packages/
      attr/
      pip/
      ...
```

Venv

```
$ python -m venv .venv

$ source .venv/bin/activate
```

https://docs.python.org/3/library/venv.html

```
$ source .venv/bin/activate

$ pip list
Package    Version
---------- -------
pip        20.0.2
setuptools 41.2.0
```

```
$ deactivate

$ pip list
Package                              Version
------------------------------------ ----------
apns                                 2.0.1
asn1crypto                           0.24.0
...
```

Как сделать свой пакет?

setup.py

```python
import setuptools

with open("README.md", "r") as fh:
    long_description = fh.read()

setuptools.setup(
    name="example-pkg-your-username",
    version="0.0.1",
    author="Example Author",
    author_email="author@example.com",
    description="A small example package",
    long_description=long_description,
    long_description_content_type="text/markdown",
    url="https://github.com/pypa/sampleproject",
    packages=setuptools.find_packages(),
)
```

https://packaging.python.org/tutorials/packaging-projects/

Имея setup.py, мы можем

```
# установить пакет
# с указанием пути
# до директории с setup.py
$ pip install .
```

Имея setup.py, мы можем

```
# установить пакет
# из внешней vcs,
# например с github
$ pip install \
-e git+https://git.repo/some_pkg.git#egg=SomeProject
```

Имея setup.py, мы можем

```
# собираем архив с пакетом
$ python setup.py sdist bdist_wheel
```

twine

```
# ставим дополнительную утилиту для сборки
$ pip install twine
...

$ twine upload dist/*  # заливаем пакет в реестр
Uploading distributions to https://pypi.org/
Enter your username: [your username]
Enter your password:
Uploading example_pkg_your_username-0.0.1-py3-none-any.whl
100%|└────────────────────┘| 4.65k/4.65k [00:01<00:00, 2.88kB/s]
Uploading example_pkg_your_username-0.0.1.tar.gz
100%|└────────────────────┘| 4.25k/4.25k [00:01<00:00, 3.05kB/s]
```

Декораторы

Синтаксический сахар

```
1 @my_decorator
2 def some_func():
3     ...
4
5
6 some_func()
```

Пример декоратора

```
1 def some_func():
2     ...
3
4
5 some_func = my_decorator(some_func)
6 some_func()
```

Пример использования

```
1  def div(func):
2    def wrapper():
3      return f'<div>{func()}</div>'
4    return wrapper
5
6
7  @div
8  def hello():
9    return 'Hello'
10
```

```
In [11]: hello()
Out[11]: '<div>Hello</div>'
```

Последовательность декораторов

```
1  @div
2  @p
3  def hello():
4    return 'Hello'
5
```

```
In [5]: hello()
Out[5]: '<div><p>Hello</p></div>'
```

Ещё пример

```
1  def timeit(func):
2    def wrapper(*args, **kwargs):
3      st = time()
4      result = func(*args, **kwargs)
5      print(func.__name__, time() - st)
6      return result
7    return wrapper
8
9
10 @timeit
11 def square(x):
12   sleep(0.1)
13   return x*x
14
```

```
In [15]: square(2)
square 0.1
Out[15]: 4
```

Как написать свой декоратор

```
1 def my_decorator(func):
2     ...
```

Как написать свой декоратор

```
1 def my_decorator(func):
2     ...
3     return wrapper  # callable object
```

Как написать свой декоратор

```
1  def my_decorator(func):
2
3    def wrapper():
4      # ...
5      result = func()
6      # ...
7
8      return result
9
10   return wrapper
```

*args, **kwargs

```
1  def my_decorator(func):
2
3    def wrapper(*args, **kwargs):
4      # ...
5      result = func(*args, **kwargs)
6      # ...
7
8      return result
9
10   return wrapper
```

Проблема

```
1 @my_docorator
2 def some_func():
3   ...
4
5
6 print(some_func.__name__)  # wrapper
```

from functools import wraps

```python
from functools import wraps


def my_decorator(func):

  @wraps(func)
  def wrapper(*args, **kwargs):
    # ...
    result = func(*args, **kwargs)
    # ...

    return result

  return wrapper
```

Нет проблемы, если использовать wraps

```
1 @my_docorator
2 def some_func():
3   ...
4
5
6 print(some_func.__name__)  # some_func
```

Конфигурируемый декоратор

```
1  @logtime(0.2)
2  def square(x):
3      sleep(0.1)
4      return x*x
5
```

```
In [6]: square(2)
Out[6]: 4
```

Конфигурируемый декоратор

```
1  def logtime(max_time=0.1):
2    def decorator(func):
3
4      def wrapper(*args, **kwargs):
5        st = time()
6
7        result = func(*args, **kwargs)
8
9        executed_time = time() - st
10       if executed_time >= max_time:
11         print(...)
12
13       return result
14
15     return wrapper
16
17   return decorator
```

Генераторы

Объект, который сохраняет состояние
между вызовами


ЧТО
ТЫ
ТАКОЕ?

Пример

```
1 g = (i for i in some_list)
2
3
4 def my_generator():
5   yield 'Hello'
```

Генератор

```
1 def my_generator():
2   print('start')
3   yield 1
4   yield 2
5   yield 3
6
7   return
```

Пример использования

```
1  g1 = my_generator()



In [2]: next(g1)
start
Out[2]: 1

In [3]: next(g1)
Out[3]: 2

In [4]: next(g1)
Out[4]: 3

In [5]: next(g1)
StopIteration:
```

for ... in ...

```
1 for i in my_generator():
2   print(i)
```

```
start
1
2
3
```

Возвращаем значение при каждом вызове

```
1 def my_generator():
2   yield 1
3   return 'Hello'
4
5
6 g = my_generator()
```

```
In [7]: next(g)
Out[7]: 1

In [8]: next(g)
StopIteration: Hello
```

yield from

```python
def generator1():
  yield 1
  yield 2
  yield 3


def generator2():
  yield from generator1()
```

```
1 def generator2():
2   yield from [1, 3, 4, 5, 6]
```

```python
def generator():
    a = yield 'Hello'
    print(a)
    yield 'Goodbye'


g = generator()
```

```
In [8]: next(g)
Out[8]: 'Hello'

In [9]: g.send('Hi')
Hi
Out[9]: 'Goodbye'
```

Зачем нужны генераторы?

```python
import re


def filter_lines(filename, regex):
  with open(filename) as f:
    for line in f:
      if re.search(regex, line):
        yield line.rstrip()
```

```python
...


for line in filter_lines('my_file.txt', '^word'):
  print(line)
```

Перерыв?

Автотесты

Зачем?


Зарефакторь, напиши тесты
Проработало 5 лет - и дальше будет! Забей!

Зачем?

- Проверяют корректность и делают это быстро
- Баги всплывают раньше (bug cost)
- Изменения в код вносить проще (feature cost)

Тесты тоже код!

assert

```
1 def square(x):
2     assert isinstance(x, int), 'type error'
3     if __debug__:
4         print('debug')
5     return x * x
```

Unittests

```python
# tests/test_utils.py
import unittest

def square(x):
    return x * x

class SquareTestCase(unittest.TestCase):

    def test_square_ok(self):
        self.assertEqual(square(3), 9)

    def test_square_error(self):
        self.assertEqual(square(3), 8)
```

```
$ python -m unittest
F.
======================================================================
FAIL: test_square_error (tests.test_utils.SquareTetsCase)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "./tests/test_utils.py", line 14, in test_square_error
    self.assertEqual(square(3), 8)
AssertionError: 9 != 8

----------------------------------------------------------------------
Ran 2 tests in 0.001s

FAILED (failures=1)
```

Pytest

https://docs.pytest.org/en/latest/

Меньше шаблонного кода

```python
# tests/test_utils.py
import unittest

def square(x):
    return x * x

class SquareTetsCase(unittest.TestCase):

    def test_square_ok(self):
        self.assertEqual(square(3), 9)

    def test_square_error(self):
        self.assertEqual(square(3), 8)
```

```python
# tests/test_utils.py

def square(x):
    return x * x


def test_square_ok():
    assert square(3) == 9


def test_square_error():
    assert square(3) == 8
```

Удобные assert`ы

```
1 # tests/test_utils.py
2 import unittest
3
4
5 class TetsCase(unittest.TestCase):
6
7     def test_dict_equal(self):
8         self.assertDictEqual({'x': 1}, {'x': 2})
```

```
1 # tests/test_utils.py
2
3 def test_dict_equal():
4     assert {'x': 1} == {'x': 2}
```

Удобные assert`ы

```
1 # tests/test_utils.py
2
3 def test_dict_equal():
4     assert {'x': 1, 'y': 3, 'z': 0} == {'x': 2, 'y': 3}
```

```
    def test_dict_equal():
>       assert {'x': 1, 'y': 3, 'z': 0} == {'x': 2, 'y': 3}
E       AssertionError: assert {'x': 1, 'y': 3, 'z': 0} == {'x': 2, 'y': 3}
E         Omitting 1 identical items, use -vv to show
E         Differing items:
E         {'x': 1} != {'x': 2}
E         Left contains more items:
E         {'z': 0}
E         Use -v to get the full diff

tests/test_utils.py:4: AssertionError
```

А так же

- фикстуры
- плагины
- параметризованные тесты
- etc.

Структура типичного теста

- `Подготовка (опционально)`
- `Действие`
- `Проверка`
- Завершение (опционально)

Начнем с начала

```python
def square(x):
    return x * x

def test_square():
    assert square(2) == 4
    assert square(-2) == 4
    assert square(2.) == 4.
    assert square(0) == 0
```

```
$ pytest
==================== test session starts ====================
platform darwin -- Python 3.7.2, pytest-4.1.1, py-1.7.0, pluggy-0.8.1
rootdir: ., inifile:
collected 4 items

tests/test_utils.py ....                                   [100%]
==================== 4 passed in 0.05 seconds ==============
```

Много маленьких тестов лучше одного большого

Начнем с начала

```python
def square(x):
    return x * x

def test_square_int():
    assert square(2) == 4

def test_square_float():
    assert square(2.) == 4.

def test_square_zero():
    assert square(0) == 0

def test_square_negative_number():
    assert square(-2) == 4
```

```
$ pytest
==================== test session starts ====================
platform darwin -- Python 3.7.2, pytest-4.1.1, py-1.7.0, pluggy-0.8.1
rootdir: ., inifile:
collected 4 items

tests/test_utils.py ....                                   [100%]
==================== 4 passed in 0.05 seconds ===============
```

Изменим реализацию

```python
def square(x):
    return x ** 2  # тут!

def test_square_int():
    assert square(2) == 4

def test_square_float():
    assert square(2.) == 4.

def test_square_zero():
    assert square(0) == 0

def test_square_negative_number():
    assert square(-2) == 4
```

```
$ pytest
==================== test session starts ====================
platform darwin -- Python 3.7.2, pytest-4.1.1, py-1.7.0, pluggy-0.8.1
rootdir: ., inifile:
collected 4 items

tests/test_utils.py ....                                   [100%]
==================== 4 passed in 0.05 seconds ===============
```

Параметризованные тесты

```python
import pytest

def square(x):
    return x ** 2

@pytest.mark.parametrize(('number', 'result'), [
    (2, 4),
    (2.0, 4.0),
    (0, 0),
    (-2, 4),
])
def test_square(number, result):
    assert square(number) == result
```

```
$ pytest -v
==================== test session starts ====================
platform darwin -- Python 3.7.2, pytest-4.1.1, py-1.7.0, pluggy-0.8.1 -- ./.venv/bin/python3
cachedir: .pytest_cache
rootdir: ., inifile:
collected 4 items

tests/test_utils.py::test_square[2-4] PASSED                                    [ 25%]
tests/test_utils.py::test_square[2.0-4.0] PASSED                                [ 50%]
tests/test_utils.py::test_square[0-0] PASSED                                    [ 75%]
tests/test_utils.py::test_square[-2-4] PASSED

==================== 4 passed in 0.05 seconds ==============
```

Добавим проверку

```
1  import math
2
3  import pytest
4
5  def square(x):
6      return x ** 2
7
8  @pytest.mark.parametrize(('number', 'result'), [
9      (2, 4),
10     (2.0, 4.0),
11     (0, 0),
12     (-2, 4),
13     (math.inf, math.inf)  # +1
14 ])
15 def test_square(number, result):
16     assert square(number) == result
```

Тесты являются контрактами того, как работает наш код

Проверка исключений

```
1 import pytest
2
3 def square(x):
4     return x ** 2
5
6 def test_square_not_number():
7     with pytest.raises(TypeError):
8         square('string')
```

Подготовка данных

```
1  class User:
2    ...
3
4
5  def test_user_hello():
6    user = User('Vasya', 20)
7    assert user.hello() == 'hello'
8
9
10 def test_user_bye():
11   user = User('Vasya', 20)
12   assert user.bye() == 'bye'
```

Фикстуры

```
1  class User:
2      ...
3
4
5  @pytest.fixture()
6  def user():
7      return User('Vasya', 20)
8
9
10 def test_user_hello(user):
11     assert user.hello() == 'hello'
12
13
14 def test_user_bye(user):
15     assert user.bye() == 'bye'
```

Fixture scope

```
1 @pytest.fixture(scope='session')
2 def user():
3     return User('Vasya', 20)
```

Фикстуры для фикстур

```
1 @pytest.fixture(scope='session')
2 def parent():
3     return User('Petya', 40)
4
5 @pytest.fixture()
6 def user(parent):
7     return User('Vasya', 20, parent)
```

Before/After test actions

```python
import pytest

@pytest.fixture()
def prepare_env():
    # our before test actions
    yield
    # our after test actions


def test_env(prepare_env):
    # our test
    pass
```

Before/After test actions

```python
import pytest

@pytest.fixture()
def prepare_env():
    init_db()
    yield
    clean_db()


def test_env(prepare_env):
    # our test
    pass
```

usefixtures

```python
import pytest

@pytest.fixture()
def prepare_env():
    # our before test actions
    yield
    # our after test actions

@pytest.mark.usefixtures('prepare_env')
def test_env():
    # our test
    pass
```

autouse

```python
import pytest

@pytest.fixture(autouse=True)
def prepare_env():
    # our before test actions
    yield
    # our after test actions

def test_env():
    # our test
    pass
```

Mock

```python
from unittest.mock import MagicMock

def test_with_mock():
    # preparing
    thing = ProductionClass()
    thing.method = MagicMock(return_value=3)

    # action
    thing.method(3, 4, 5, key='value')

    # asserts
    thing.method.assert_called_with(3, 4, 5, key='value')
```

https://docs.python.org/3/library/unittest.mock.html

Зачем?

- Подменяем реальные объекты
- Задаем нужные нам возвращаемые значения/исключения
- Проверяем вызывался ли нужный метод/функция на самом деле

pytest-mock

```python
import os

class UnixFS:
    @staticmethod
    def rm(filename):
        os.remove(filename)

def test_unix_fs(mocker):
    mocker.patch('os.remove')
    UnixFS.rm('file')
    os.remove.assert_called_once_with('file')
```

https://github.com/pytest-dev/pytest-mock/

Пример

```
1 @pytest.fixture()
2 def user(mocker):
3     mock = mocker.MagicMock()
4     mock.hello.return_value = 'hello'
5     mock.bye.return_value = 'bye'
6     return mock
```

spec

```
1 @pytest.fixture()
2 def user(mocker):
3     return mocker.MagicMock(spec=User)
```

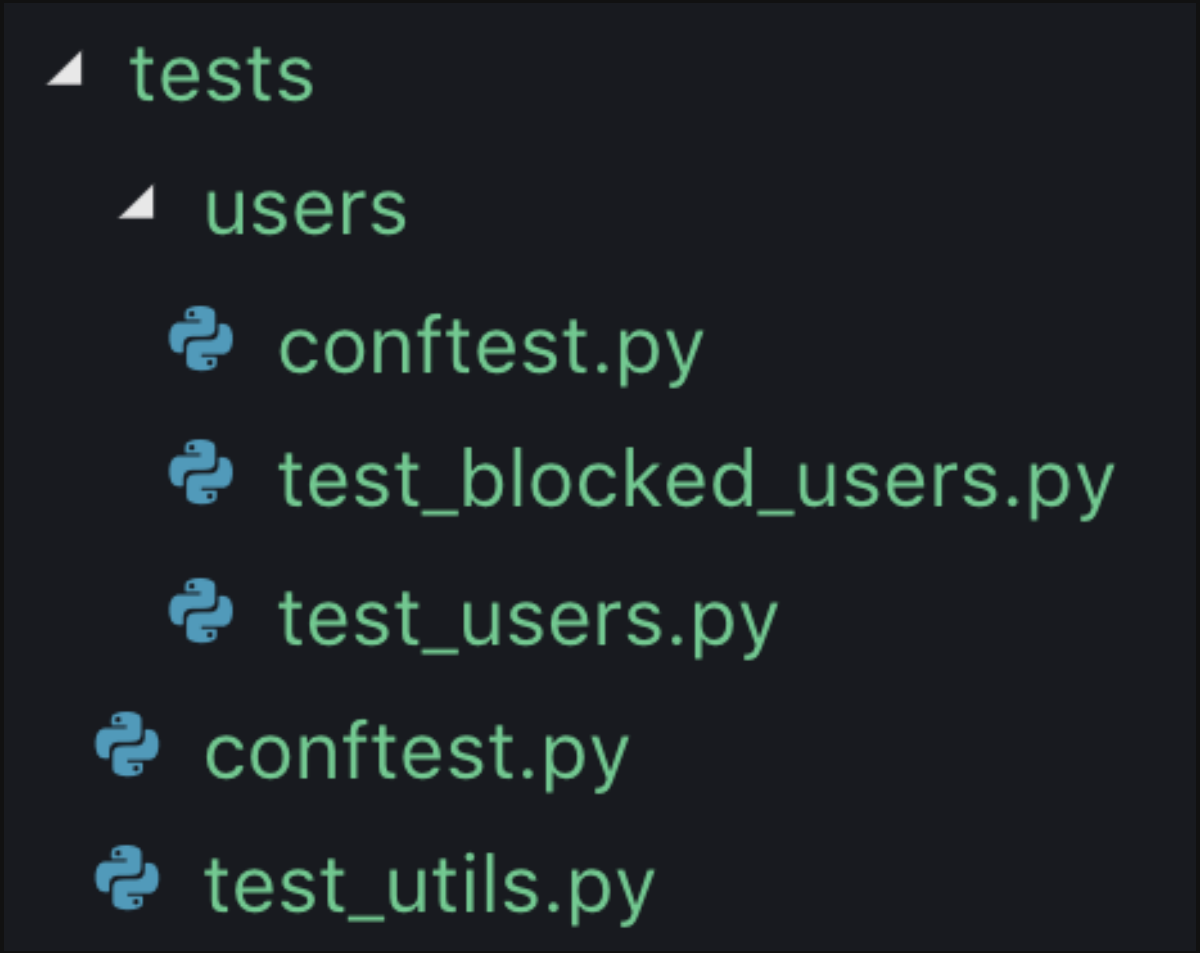
mocker.patch

```python
import requests
import pytest


def get_page():
    return requests.get('www.google.com')


@pytest.fixture()
def requests_mock(mocker):
    return mocker.patch('requests.get')


def test_get_page(requests_mock):
    requests_mock.return_value.json.return_value = {'id': 1}
    response = get_page()
    assert response.json() == {'id': 1}
```

mocker.patch.object

```
1  def test_mocked_user_1(mocker, user):
2      mock = mocker.patch.object(user, 'bye')
3      mock.return_value = 'hello'
4      assert user.bye() == 'hello'
5
6
7  def test_mocked_user_2(mocker, user):
8      mocker.patch.object(user, 'bye', return_value='hello')
9      assert user.bye() == 'hello'
10
11
12 def test_mocked_user_3(mocker, user):
13     mocker.patch.object(user, 'bye').return_value = 'hello'
14     assert user.bye() == 'hello'
```

conftest.py


tests
users
conftest.py
test_blocked_users.py
test_users.py
conftest.py
test_utils.py

Code coverage

Pytest-cov

```
$ pytest --cov=app tests/
...


---------------------- coverage: ... ----------------------
Name                    Stmts   Miss  Cover
-----------------------------------------
app/__init__             2       0     100%
app/__main__             257     13    94%
app/utils                94      7     92%
-----------------------------------------
TOTAL                    353     20    94%
```

https://github.com/pytest-dev/pytest-cov

Debugger

breakpoint etc.

Debugger

```python
def test_dict_equal():
    ... # many code lines
    breakpoint()
    assert {'x': 1, 'y': 3, 'z': 0} == {'x': 2, 'y': 3}
```

```
>>>>>>> PDB set_trace (IO-capturing turned off) >>>>>>>>>>>>>
> ./tests/test_utils.py(6)test_dict_equal()
-> assert {'x': 1, 'y': 3, 'z': 0} == {'x': 2, 'y': 3}
(Pdb) h

Documented commands (type help <topic>):
========================================
EOF     cl         display  interact  n      restart  step       up
a       clear      down     j         next   return   tbreak     w
...
```

Debugger

```
$ pytest --pdb tests
```

PEPs

Python Enhancement Proposal

PEP8

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

Структура проекта

README.md

```
1 # My project
2
3 Descriptions bla-bla-bla...
4
5 ## Run tests
6
7 ```
8 make test
9 ```
```

README.rst

pytest-flask

pypi v0.14.0 | conda-forge v0.14.0 | python 2.7 | 3.4 | 3.5 | 3.6 | 3.7 | docs passing

An extension of pytest test runner which provides a set of useful tools to simplify testing and development of the Flask extensions and applications.

To view a more detailed list of extension features and examples go to the PyPI overview page or package documentation.

How to start?

Define your application fixture in `conftest.py`:

```
from myapp import create_app

@pytest.fixture
def app():
    app = create_app()
    return app
```

Install the extension with dependencies and go:

```
$ pip install pytest-flask
$ pytest
```

Makefile

https://www.gnu.org/software/make/manual/make.html

Makefile

```
1  PYTHONPATH = PYTHONPATH=./
2
3  TEST = $(PYTHONPATH) pytest --verbosity=2 ... $(arg)
4  CODE = tests app
5
6
7  .PHONY: test test-failed
8
9  test:
10   $(TEST) --cov
11
12 test-failed:
13   $(TEST) --last-failed
```

.gitignore

https://git-scm.com/docs/gitignore

В итоге типичный проект будет выглядеть как-то так

```
app/
   __init__.py
   __main__.py
   utils.py
   ...
tests/
   conftest.py
   test_utils.py
   ...
 .gitignore
 Makefile
 requirements.txt # or setup.py
 README.md
```

ДЗ