pytest overview

It's a framework

Unittest --> pytest

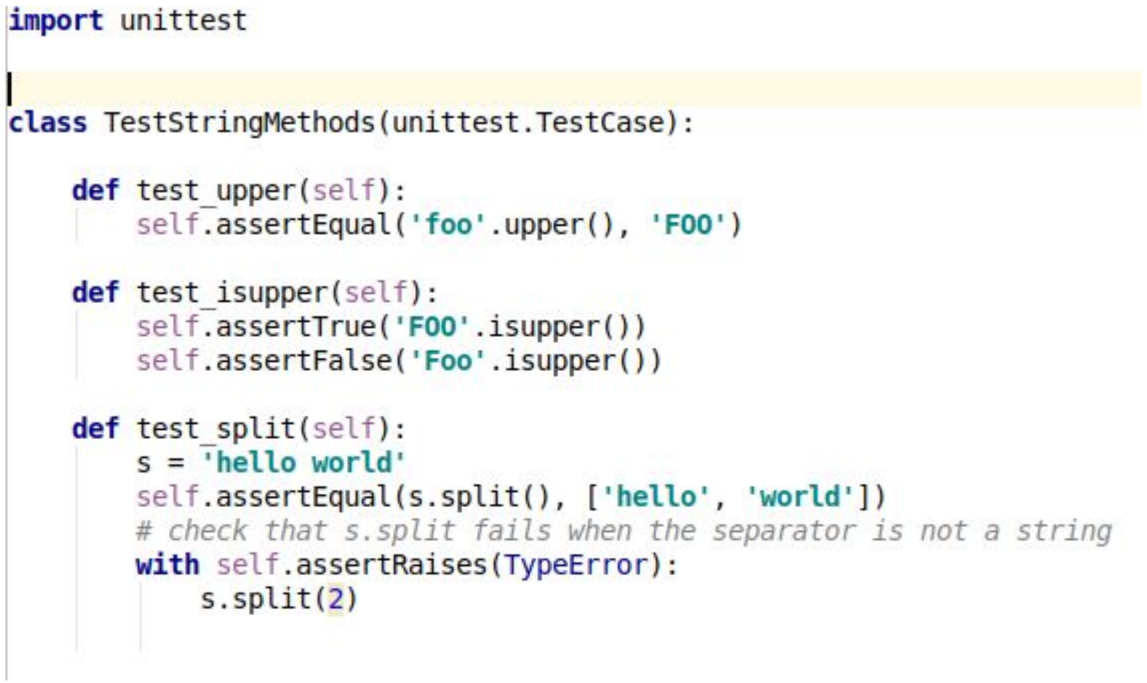
Start

It is possible to run unittest-based tests with pytest.

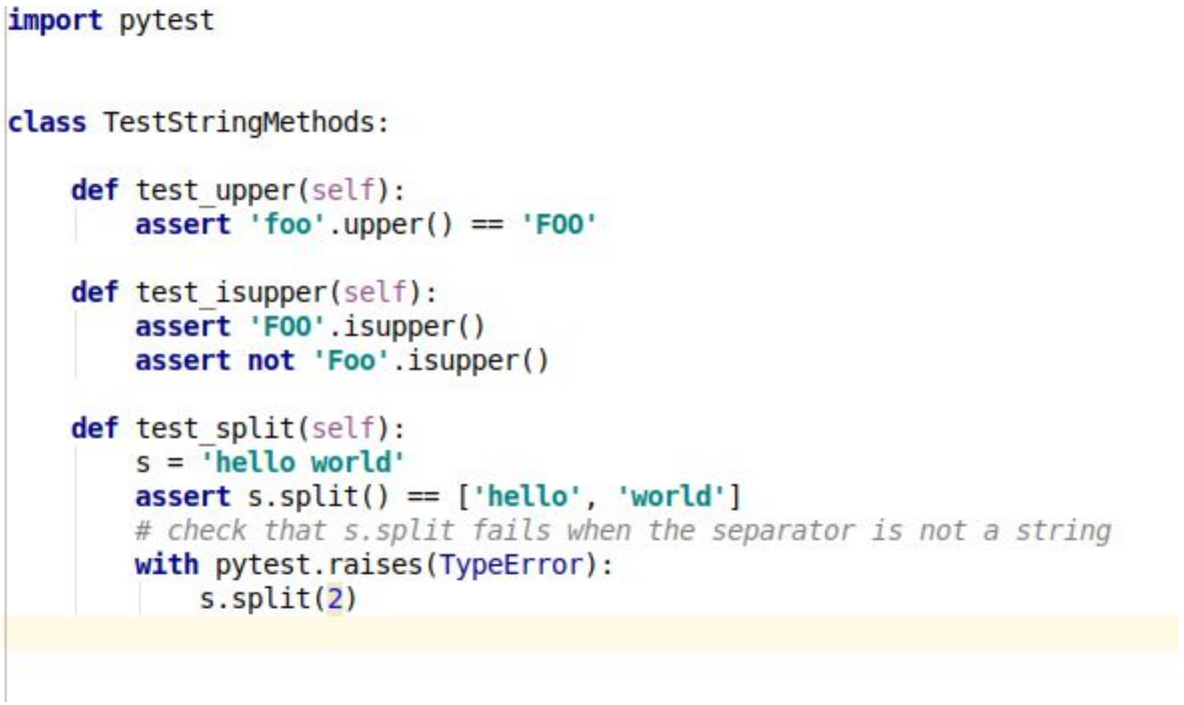
```
$ python -m unittest
```

```
$ pytest
```

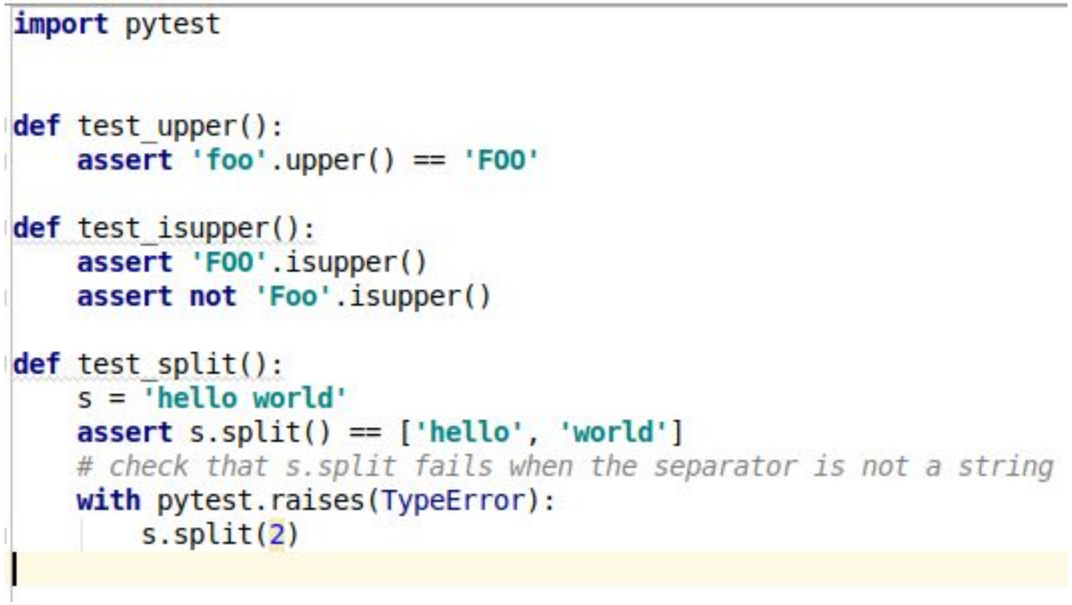
Less boilerplate



```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)
```



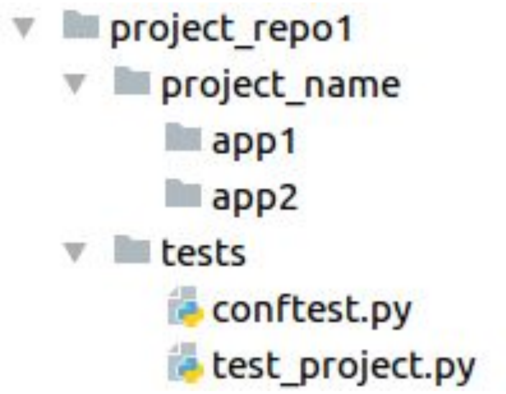
```python
import pytest


class TestStringMethods:

    def test_upper(self):
        assert 'foo'.upper() == 'FOO'

    def test_isupper(self):
        assert 'FOO'.isupper()
        assert not 'Foo'.isupper()

    def test_split(self):
        s = 'hello world'
        assert s.split() == ['hello', 'world']
        # check that s.split fails when the separator is not a string
        with pytest.raises(TypeError):
            s.split(2)
```



```python
import pytest


def test_upper():
    assert 'foo'.upper() == 'FOO'

def test_isupper():
    assert 'FOO'.isupper()
    assert not 'Foo'.isupper()

def test_split():
    s = 'hello world'
    assert s.split() == ['hello', 'world']
    # check that s.split fails when the separator is not a string
    with pytest.raises(TypeError):
        s.split(2)
```

Project structure

Basic

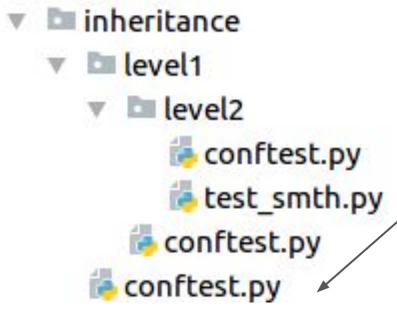


```
▼ project_repo1
  ▼ project_name
      app1
      app2
  ▼ tests
      conftest.py
      test_project.py
```

auto-discovery

composition over inheritance

Implicit conftest inheritance


```
▼ inheritance
    ▼ level1
        ▼ level2
            conftest.py
            test_smth.py
        conftest.py
    conftest.py
```

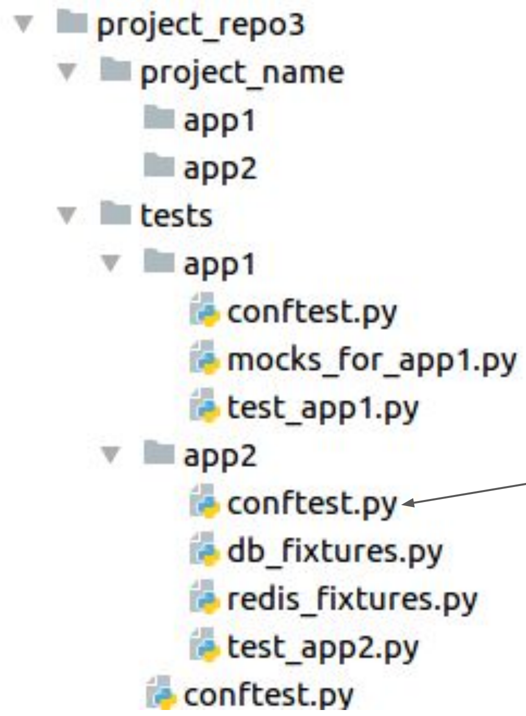
```python
import pytest

@pytest.fixture
def fx_from_level0():
    return 3
```

```python
import pytest

@pytest.fixture
def fx_from_level1(fx_from_level0):
    return fx_from_level0 * 10
```

```python
import pytest

@pytest.fixture
def fx_from_level2(fx_from_level1):
    return fx_from_level1 * 2
```

```python
def test_smth(fx_from_level2):
    assert fx_from_level2 == 3 * 10 * 2
```

Modules

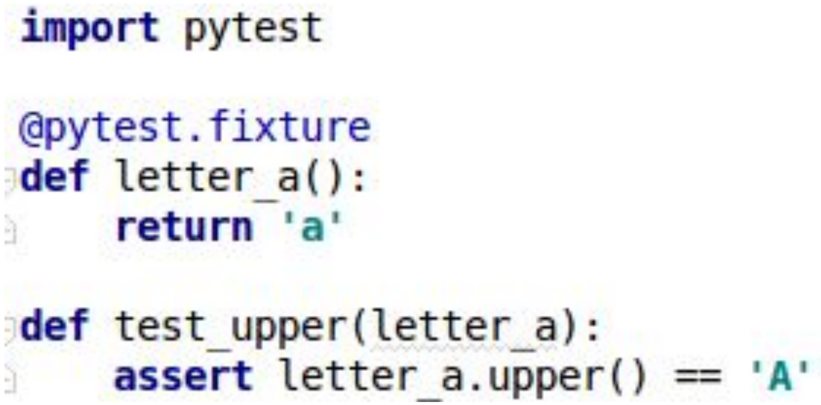


```
project_repo3
  project_name
    app1
    app2
  tests
    app1
      conftest.py
      mocks_for_app1.py
      test_app1.py
    app2
      conftest.py
      db_fixtures.py
      redis_fixtures.py
      test_app2.py
    conftest.py
```

```
# conftest.py

pytest_plugins = ['db_fixtures', 'redis_fixtures']
```

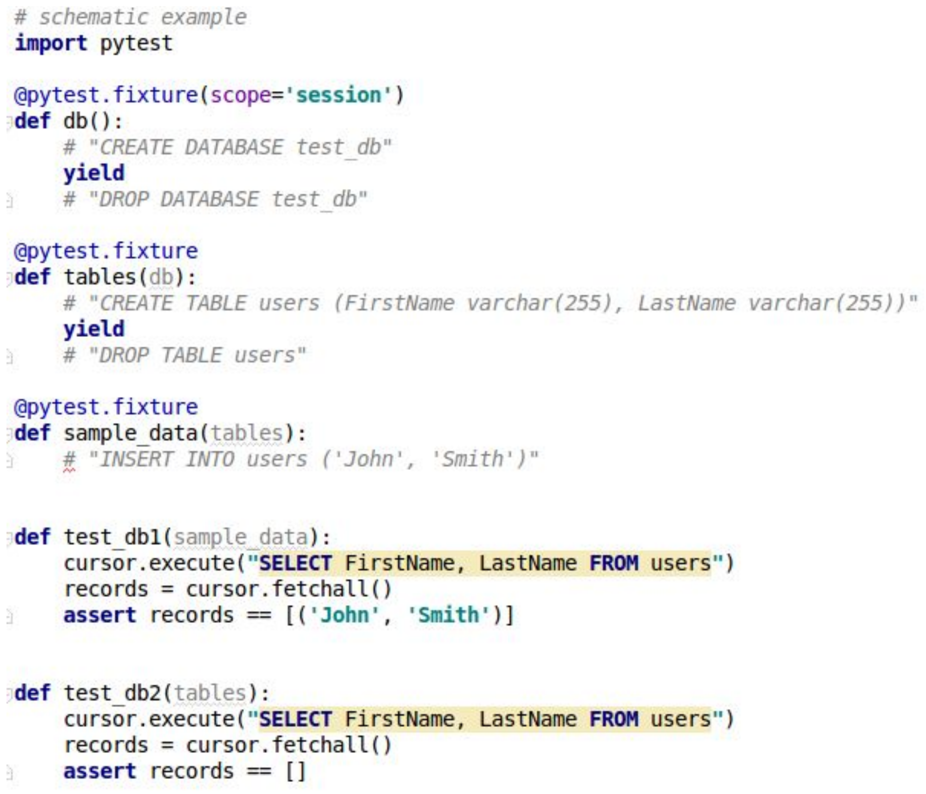
Fixtures

Simple fixture



```
import pytest

@pytest.fixture
def letter_a():
    return 'a'

def test_upper(letter_a):
    assert letter_a.upper() == 'A'
```

Setup/Teardown via `yield`



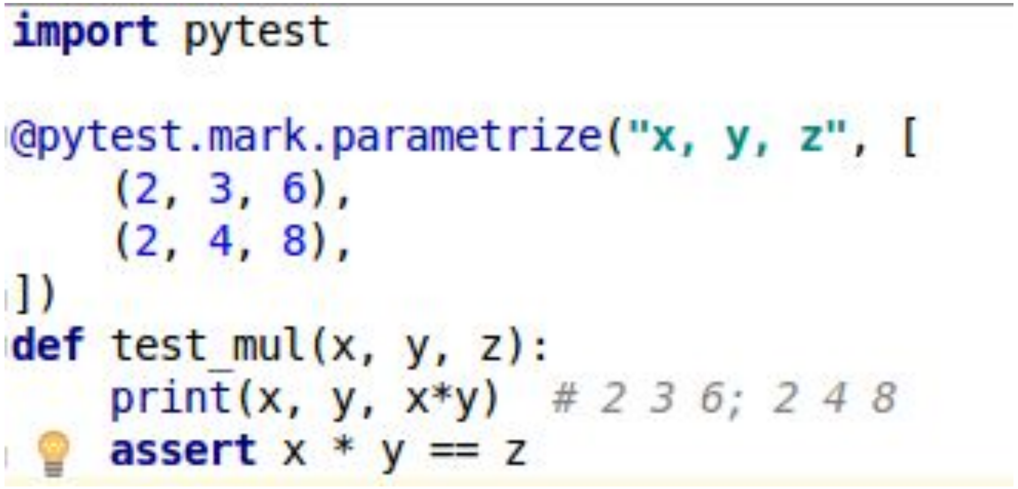
```python
# schematic example
import pytest

@pytest.fixture(scope='session')
def db():
    # "CREATE DATABASE test_db"
    yield
    # "DROP DATABASE test_db"

@pytest.fixture
def tables(db):
    # "CREATE TABLE users (FirstName varchar(255), LastName varchar(255))"
    yield
    # "DROP TABLE users"

@pytest.fixture
def sample_data(tables):
    # "INSERT INTO users ('John', 'Smith')"


def test_db1(sample_data):
    cursor.execute("SELECT FirstName, LastName FROM users")
    records = cursor.fetchall()
    assert records == [('John', 'Smith')]


def test_db2(tables):
    cursor.execute("SELECT FirstName, LastName FROM users")
    records = cursor.fetchall()
    assert records == []
```

Fixture scope

- session
- module
- class
- function [default]
- package scope (experimental)

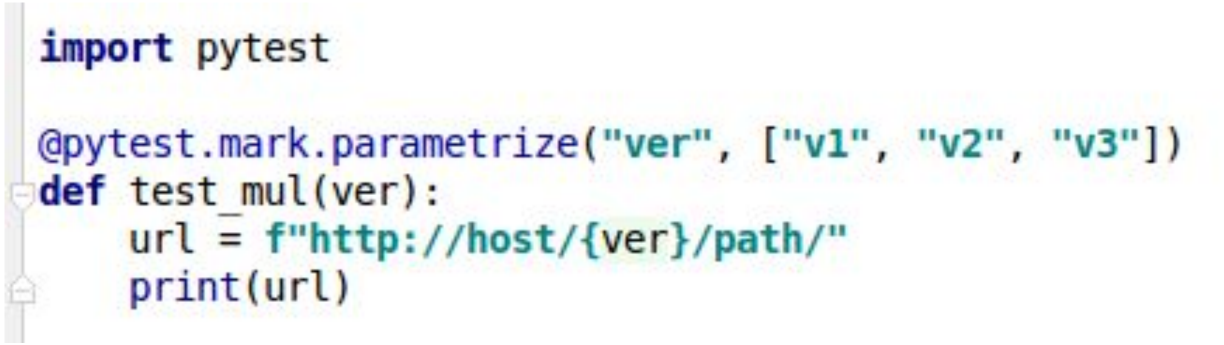
```
@pytest.fixture(scope="module")
def test_smth():
    ...
```

Parameterization

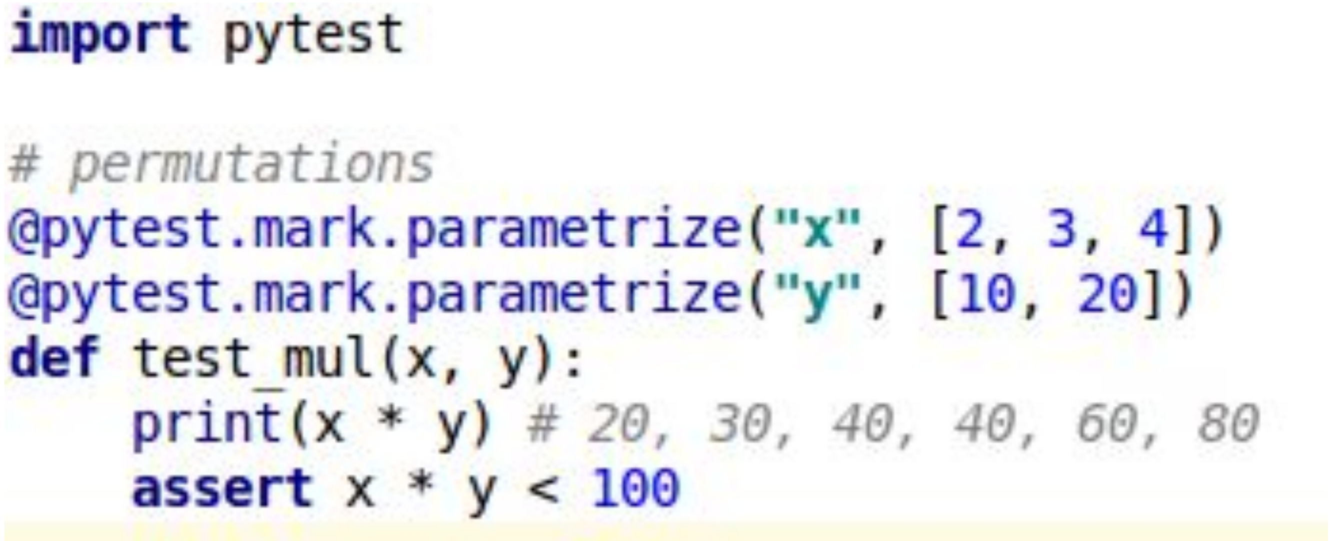


```
import pytest

@pytest.mark.parametrize("x, y, z", [
    (2, 3, 6),
    (2, 4, 8),
])
def test_mul(x, y, z):
    print(x, y, x*y)  # 2 3 6; 2 4 8
    assert x * y == z
```

API versions

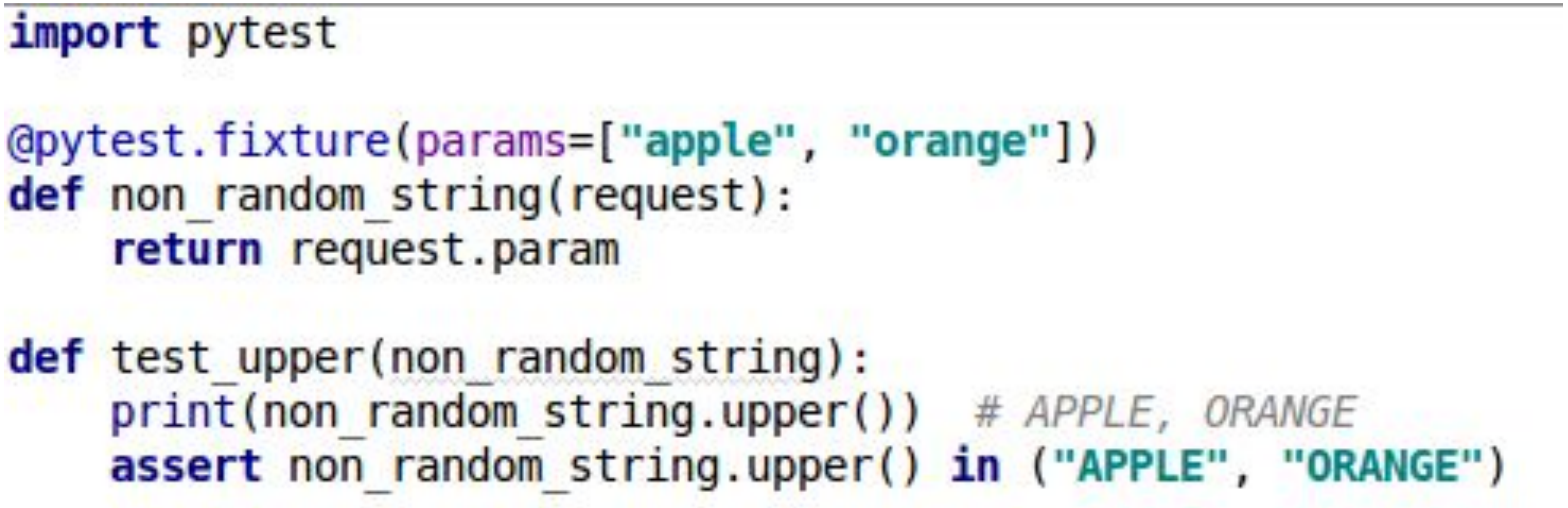


```python
import pytest

@pytest.mark.parametrize("ver", ["v1", "v2", "v3"])
def test_mul(ver):
    url = f"http://host/{ver}/path/"
    print(url)
```



```python
import pytest

# permutations
@pytest.mark.parametrize("x", [2, 3, 4])
@pytest.mark.parametrize("y", [10, 20])
def test_mul(x, y):
    print(x * y) # 20, 30, 40, 40, 60, 80
    assert x * y < 100
```

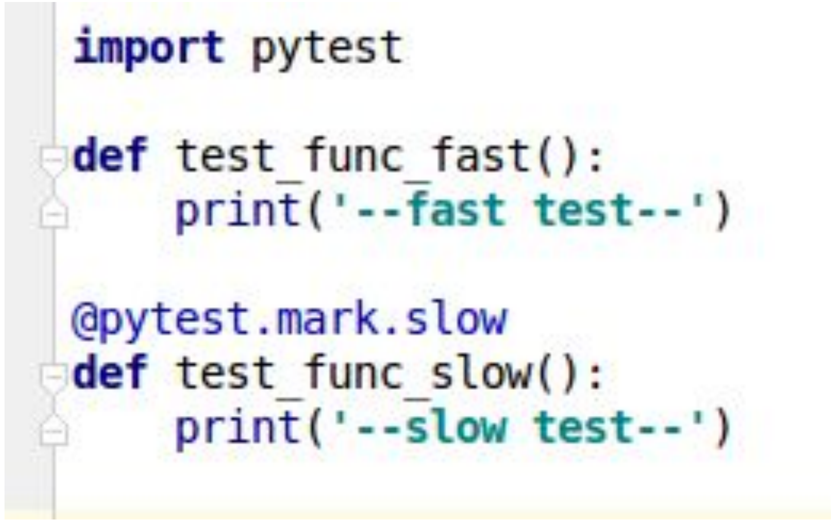
Parameterized fixtures



```python
import pytest

@pytest.fixture(params=["apple", "orange"])
def non_random_string(request):
    return request.param

def test_upper(non_random_string):
    print(non_random_string.upper())  # APPLE, ORANGE
    assert non_random_string.upper() in ("APPLE", "ORANGE")
```

Slow tests

pytest -m "not slow" # only run tests matching given mark expression



```python
import pytest

def test_func_fast():
    print('--fast test--')

@pytest.mark.slow
def test_func_slow():
    print('--slow test--')
```

Skip tests

```python
@pytest.mark.skip(reason="no way of currently testing this")
def test_the_unknown():
    ...
```

```python
import sys
@pytest.mark.skipif(sys.version_info < (3,6),
                    reason="requires python3.6 or higher")
def test_function():
    ...
```

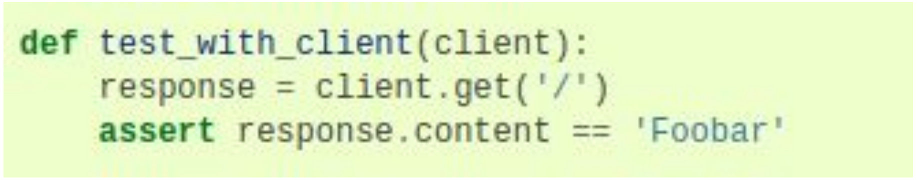
Plugins

Frameworks

pytest-django
  *-flask, -aiohttp, -twisted

pytest-sanic (requires `aiohttp` ¯_(ツ)_/¯)

pytest-django



```
def test_with_client(client):
    response = client.get('/')
    assert response.content == 'Foobar'
```

- useful fixtures
- database creation/re-use

https://pytest-django.readthedocs.io/en/latest/helpers.html#fixtures

pytest-xdist

Test run parallelization:
multiple CPUs, remote hosts, subprocesses etc

Note:
You may not need it.
- avoid slow tests
- let CI care about them
- $ django-admin test --parallel [N]
- async approach is different

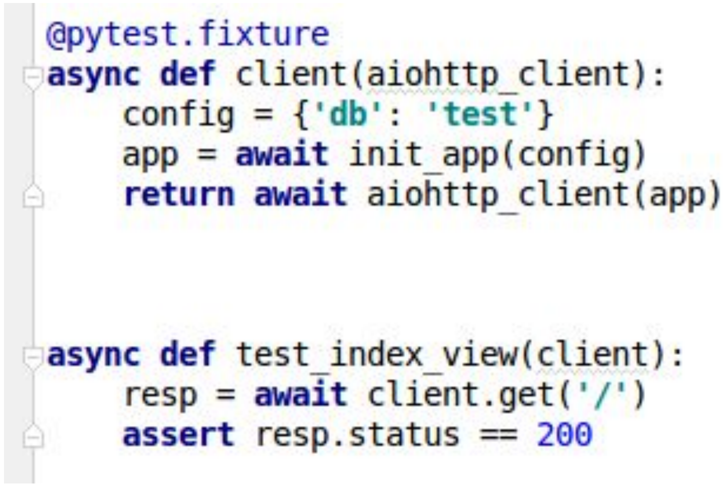
pytest-cov

pytest-socket

... many more

Asyncio

- pytest-asyncio vs pytest-aiohttp
- async tests are ok



```python
@pytest.fixture
async def client(aiohttp_client):
    config = {'db': 'test'}
    app = await init_app(config)
    return await aiohttp_client(app)


async def test_index_view(client):
    resp = await client.get('/')
    assert resp.status == 200
```

Command line

```
$ pytest tests/test_file.py::TestClass::test_method          # run specific method

-s                                        # do not capture output (like prints)

-q, --quiet                               # decrease verbosity

$ pytest --fixtures test_file.py          # show available fixtures

-m MARKEXPR                               # only run tests matching given mark expression. E.g.
$ pytest -m slow                          # run tests decorated with @pytest.mark.slow

--pdb                                     # drop into pdb on failure

$ pytest -k "MyClass and not method"      # by keyword expressions
```

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

Configuration

- command line
- pytest.ini
- setup.cfg
- tox.ini ([pytest])

Nuances

- PYTHONPATH

  python -m pytest [...]

- custom django settings configuration (e.g. modified manage.py)

  Solution depends. Configure in root `conftest.py`, use `--ds` option etc

- pytest -o addopts= tests   # to ignore options from tox.ini

Useful links

Productive pytest with PyCharm
https://www.youtube.com/watch?v=ixqeebhUa-w

Продвинутое использование py test, Андрей Светлов, Python Core Developer
https://www.youtube.com/watch?v=7KgihdKTWY4

Thanks.