به نام خدا

PyTest | Amir Abbas | IUT | Dr.Mahmoudzadeh

Testing in Python?

- easy to write
- easy to read
- easy to run
- memory & CPU are not problems
- deployment is not a problem

What is py.test?

- open-source Python testing tool
- "helps you write better programs"
- not included in stdlib but very well-known
- active since 2007 and actively developed

```
pip install pytest
easy_install pytest
```

Write and run your first test

```
# Function to test
def division(a, b):
    return a / float(b)
```

1. create a function with the test_ prefix

```
def test_division():
    assert division(1, 1) == 1
```

2. run tests

```
py.test division.py
```

Passing test

```
MacBookPro:Example_1 Jean$ py.test division.py
============================= test session starts ==============================
platform darwin -- Python 2.7.10, pytest-2.8.3, py-1.4.31, pluggy-0.3.1
rootdir: /Users/Jean/Dropbox/pytest-intro/Example_1, inifile:
collected 1 items

division.py .

=========================== 1 passed in 0.01 seconds ===========================
```

Failing test

```
MacBookPro:Example_1 Jean$ py.test division.py
============================= test session starts ==============================
platform darwin -- Python 2.7.10, pytest-2.8.3, py-1.4.31, pluggy-0.3.1
rootdir: /Users/Jean/Dropbox/pytest-intro/Example_1, inifile:
collected 1 items

division.py F

=================================== FAILURES ===================================
_____________________________ test_division ______________________________

    def test_division():
>       assert division(1, 1) == 2
E    assert 1.0 == 2
E     +  where 1.0 = division(1, 1)

division.py:8: AssertionError
=========================== 1 failed in 0.01 seconds ===========================
```

How are tests discovered?

Any way you can imagine!!!

```
py.test
```

```
py.test test_module.py
```

```
py.test test_module.py::test_function
```

```
py.test test_module.py -k my_keyword
```

Plugins

- web frameworks (django, flask, ...)
- code coverage
- xdist: multicore test distribution
- timeout: long tests management
- pep8
- ... (150+ plugins)

No plugin, write it!

- good documentation
- hooks for all steps

Other modules?

`unittest`

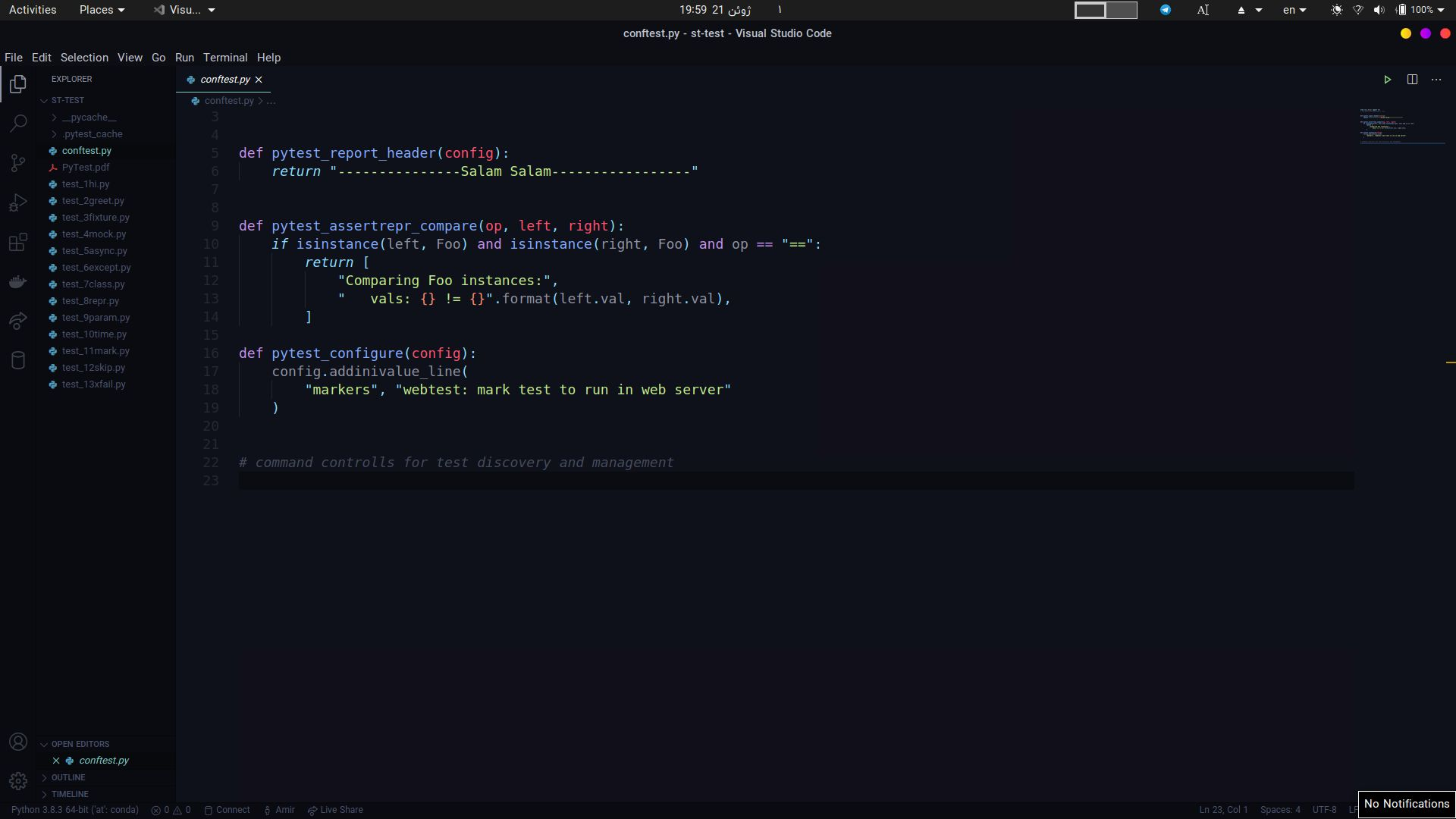
- included in stdlib
- easy to use
- lot of code to write
- limited

`nose`

- advanced features compared to unittest
- more configuration needed
- no fixtures

So, why py.test?

- runs unittest and nose tests
- easy
- advanced features
- plugins
- runs on everything
- good documentation
- active development
- used by Fedora, Mozilla, ...


EXPLORER
ST-TEST
__pycache__
.pytest_cache
conftest.py
PyTest.pdf
test_1hi.py
test_2greet.py
test_3fixture.py
test_4mock.py
test_5async.py
test_6except.py
test_7class.py
test_8repr.py
test_9param.py
test_10time.py
test_11mark.py
test_12skip.py
test_13xfail.py
conftest.py
conftest.py > ...
def pytest_report_header(config):
return "--------------Salam Salam----------------"
def pytest_assertrepr_compare(op, left, right):
if isinstance(left, Foo) and isinstance(right, Foo) and op == "==":
return [
"Comparing Foo instances:",
" vals: {} != {}".format(left.val, right.val),
]
def pytest_configure(config):
config.addinivalue_line(
"markers", "webtest: mark test to run in web server"
)
command controlls for test discovery and management
OPEN EDITORS
conftest.py
OUTLINE
TIMELINE

```python
def test_pass():
    assert 'hello tested world' == 'hello' + ' tested world'

def test_fail():
    assert 1 == 2
```

EXPLORER

ST-TEST

- __pycache__
- .pytest_cache
- conftest.py
- PyTest.pdf
- test_1hi.py
- test_2greet.py
- test_3fixture.py
- test_4mock.py
- test_5asyncp.py
- test_6except.py
- test_7class.py
- test_8repr.py
- test_9param.py
- test_10time.py
- test_11mark.py
- test_12skip.py
- test_13xfail.py

test_2greet.py

test_2greet.py > test_greet

```python
def greet(person):
    return 'Hi {name}'.format(**person)

def test_greet():
    alice = {'name': 'Alice'} # Arrange
    greeting = greet(alice) # Act
    assert greeting == "Hi Alice" # Assert

"""
Arrange: Setup any variables or conditions your test needs.
Act: Execute the code you want to test.
Assert: Check that the code behaviours in a way that you would expect
"""
```

OPEN EDITORS

- test_2greet.py

OUTLINE

TIMELINE

```python
@pytest.fixture(scope="function", params=[1, 2])
def otherarg(request):
    param = request.param
    print("  SETUP otherarg", param)
    yield param
    print("  TEARDOWN otherarg", param)


def test_0(otherarg):
    print("  RUN test0 with otherarg", otherarg)


def test_1(modarg):
    print("  RUN test1 with modarg", modarg)


def test_2(otherarg, modarg):
    print("  RUN test2 with otherarg {} and modarg {}".format(otherarg, modarg))

```

```python
from unittest import mock
from random import randint


def sum_with_rand(number):
    return number + randint(1, 10)


@mock.patch(__name__+".randint", return_value=3, autospec=True)
def test_sum_with_rand(mock_randint):
    assert sum_with_rand(2) == 5

```

test_5async.py

```python
import pytest
async def get():
    return 1 + 2


@pytest.mark.asyncio
async def test_get():
    assert await get() == 3
```

```python
import pytest


def f():
    raise SystemExit(1)


def test_mytest():
    with pytest.raises(SystemExit):
        f()
```

```python
class TestClass:
    def test_one(self):
        x = "this"
        assert "h" in x

    def test_two(self):
        x = "hello"
        assert hasattr(x, "check")
```

```
3  class Foo:
4      def __init__(self, val):
5          self.val = val
6
7      def __eq__(self, other):
8          return self.val == other.val
9
10
11 def test_compare():
12     f1 = Foo(1)
13     f2 = Foo(2)
14     assert f1 == f2
```

```python
import pytest


@pytest.mark.parametrize("test_input,expected", [("3+5", 8), ("2+4", 6), ("6*10", 8)])
def test_eval(test_input, expected):
    assert eval(test_input) == expected
```

```python

def test_funcfast():
    time.sleep(0.1)


def test_funcslow1():
    time.sleep(0.2)


def test_funcslow2():
    time.sleep(0.3)

# pytest --durations=3 test_10time.py
```

```python
import pytest


@pytest.mark.webtest
def test_send_http():
    pass  # perform some webtest test for your app


def test_something_quick():
    pass


def test_another():
    pass


class TestClass:
    def test_method(self):
        pass

# pytest test_11mark.py -m webtest
```

test_12skip.py

```python
import sys
import pytest


@pytest.mark.skipif()
def test_skipped():
    assert 1


@pytest.mark.skipif(sys.version_info[0] == 3, reason="Only Python 2")
def test_skipped_if():
    assert 1

```

test_13xfail.py

```python
import pytest


def division(a, b):
    try:
        return a / float(b)
    except ZeroDivisionError:
    # except RecursionError:
        return 1


@pytest.mark.xfail
def test_division_fail():
    assert division(1, 0) == 1

```

Resources

- https://slides.com/jeancruypenynck/introduction-to-pytest
- https://docs.pytest.org/en/stable/
- https://www.guru99.com/

Thanks