Mastering Pytest

@__mharrison__

Assignment

Assignment 1

METASNAKE

Assignment

Assignment 2

METASNAKE

Test Parameterization

Test Parameterization

```
@pytest.mark.parametrize('x, y, z', [
                (1, 2, 3), # first
                (-1, -2, -3), # neg
                (0, 0, 0)]) # test 0s
def test_add2(x, y, z):
    assert adder(x, y) == z
```

Test Parameterization

Note that the Node Ids change:

```
$ PYTHONPATH=./ pytest tests/*.py -v
================ test session starts =================
platform darwin -- Python 3.6.4, pytest-3.0.6, py-1.4.32, pluggy-0.4.0 --
/Users/matt/.env/36/bin/python3
cachedir: .cache
rootdir: /Users/matt/code_samples/pytest/Project, inifile:
plugins: asyncio-0.8.0
collected 5 items

tests/test_adder.py::test_ints PASSED
tests/test_adder.py::test_big SKIPPED
tests/test_adder.py::test_add2[1-2-3] PASSED
tests/test_adder.py::test_add2[-1--2--3] PASSED
tests/test_adder.py::test_add2[0-0-0] PASSED

======== 4 passed, 1 skipped in 0.03 seconds =========
```

Assignment

Assignment 3

METASNAKE

Fixtures

Fixtures

Provides consistent tests by dependency injection and setup/teardown

Fixtures

```
@pytest.fixture
def large_num():
    return 1e20

def test_large(large_num):
    assert adder(large_num, 1) ==
large_num
```

Method Fixtures

```
class TestAdder:

    @pytest.fixture
    def other_num(self):
        return 42

    def test_other(self, other_num):
        assert adder(other_num, 1) == 43
```

Fixtures Parameterization

```python
@pytest.fixture(params=[-1, 0, 100])
def num(request):
    return request.param

def test_num(num):
    assert adder(num, 1) == num + 1
```

See Fixtures

Run:

```
$ pytest --fixtures

----- fixtures defined from pytest_cov.plugin -----
cov
    A pytest fixture to provide access to the underlying
coverage object.

----- fixtures defined from test_funcs_pytest -----
a
    tests/test_funcs_pytest.py:17: no docstring available
```

Assignment

Assignment 4

METASNAKE

More Fixtures

Teardown in Fixtures

- Use `setup/teardown`
- Use `request` fixture and call `request.addfinalizer(fn)`
- Use generator

Module Level

Called before and after every function in the module is called:

```
def setup_module():
    ...

def teardown_module():
    ...
```

Class Level

Called before and after all test methods of class:

```
class TestFoo:
    @classmethod
    def setup_class(cls):
        ...

    @classmethod
    def teardown_class(cls):
        ...
```

Method Level

Called before and after every method:

```
class TestFoo:
    def setup_method(self):
        ...

    def teardown_method(self):
        ...
```

Function Level

Called before and after every function:

```
def setup_function():
    ...

def teardown_function():
    ...
```

`request`

Some parts of the request content:

- `r.addfinalizer(f)` - call when done
- `r.applymarker(m)` - dynamically add marker
- `r.config` - pytest config
- `r.keywords` - keywords and markers
- `r.param` - value of parameterization

Finalizer

```
@pytest.fixture
def db_num(request):
    # connect to db
    num = db.get()
    def fin():
        db.close()
    request.addfinalizer(fin)
    return num
```

Note - can have more than one finalizer function

Generator

```
@pytest.fixture
def db_num():
    # connect to db
    num = db.get()
    yield num
    db.close()
```

Generator

Code smell:

```
from contextlib import closing

@pytest.fixture
def db_num():
    # connect to db
    with closing(get_db()) as db:
        num = db.get()
        yield num
```

Fixture Scope

- `session` - Once per test session
- `module` - Once per module
- `class` - Once per test class
- `function` - Once per test function (default)

Fixture Scope

```
@pytest.fixture(
     scope='session')
def start_time():
     import time
     return time.time()
```

Fixture Scope

```
@pytest.fixture(
    scope='session')
def session_db():
    db = get_db()
    yield db
    db.close()
```

Fixture Scope

```
from contextlib import closing

@pytest.fixture(
    scope='session')
def session_db():
    with closing(get_db()) as db:
        yield db
```

Fixture Scope

Finer grained scope can depend on larger grain, but reverse is not true

Fixture Scope

```
# bad fixture depend
@pytest.fixture(scope='function')
def two():
    return 2

@pytest.fixture(scope='session')
def four(two):
    return two * two

def test4(four):
    assert four == 4
```

Fixture Scope

```
================== ERRORS ==========================
________ ERROR at setup of test4 ___________________
ScopeMismatch: You tried to access the 'function'
scoped fixture 'two' with a 'session' scoped
request object, involved factories
tests/test_adder.py:45:  def four(two)
tests/test_adder.py:41:  def two()
== 6 passed, 1 skipped, 1 error in 0.03 seconds ===
```

Trigger skip from fixture

```
@pytest.fixture
def db_num(request):
    # connect to db
    try:
        num = db.get()
        return num
    except ConnectionError:
        pytest.skip("No DB")
```

Pass data from marks to fixtures

```
@pytest.fixture
def db_con(request):
    name =
request.node.get_marker('pg_db').args[0]
    return psycopg2.connect("dbname={}".format(
        name))

@pytest.mark.pg_db('test')
def test_pg(db_con):
    # select from test db
```

Skip tests on Mac

Use `autouse=True` to implicitly enable

```
@pytest.mark.nomac
def test_add_nomac():
    # ...

@pytest.fixture(autouse=True)
def skip_mac(request):
    mark = request.node.get_marker('nomac')
    if mark and sys.platform == 'darwin':
        pytest.skip('Skip on Mac')
```

Assignment

Assignment 5

METASNAKE

Monkey Patch Fixture

Monkey Patch

Builtin fixture `monkeypatch` can:

- `chdir` - change current working directory
- `delattr` - remove attribute
- `delenv` - remove environment variable
- `delitem` - remove via index operation
- `setattr` - set attribute
- `setenv` - set environment variable
- `setitem` - set with index operation
- `syspath_prepend` - insert path into `sys.path`

Monkey Patch

```
def test_mp(monkeypatch):
    from proj import adder
    def new_add(x, y):
        return x - y
    monkeypatch.setattr(adder, 'adder',
new_add)
    assert adder.adder(1,3) == -2
```

Assignment

Assignment 6

METASNAKE

Configuration

Configuration

- Rootdir
  - Node ids determined from root
  - Plugins may store data there
- `pytest.ini` (or `tox.ini` or `setup.cfg`)
  - Must have `[pytest]` section

Some INI Options

Run to get all of `pytest.ini` settings:

```
$ pytest --help
```

Some INI Options

- `minversion = 4.0` - Fail if pytest < 4.0
- `addopts = -v` - Add verbose flag
- `norecursedirs = .git` - Don't look in `.git` directory
- `testpaths = regression` - Look in `regression` folder if no locations specified on command
- `python_files = regtest_*.py` - Execute files starting with `regtest_` (`test_*.py` and `*_test.py` default)
- `python_classes = RegTest*` - Use class starting with `RegTest` as a test (default `Test*`)
- `python_functions = *_regtest` - Use function ending with `regtest` as test (default `_test`)

METASNAKE

Conftest

Can create a `conftest.py` in a root directory or subdirectory. You can put fixtures in here. You don't import this module. Pytest loads it for you

Assignment

Assignment 7

METASNAKE

Plugins

Plugins

You can have local plugins and installable plugins

Many Hooks

- Bootstrap - for `setup.py` plugins
- Initialization hooks - for `conftest.py`
- `runtest` hooks - for execution
- Collection hooks
- Reporting hooks
- Debugging hooks

Examples

- `pytest_addoption(parser)`
- `pytest_ignore_collect(path, config)`
- `pytest_sessionstart(session)`
- `pytest_sessionfinish(session, exitstatus)`
- `pytest_assertrepr_compare(config, op, left, right)`

https://docs.pytest.org/en/latest/writing_plugins.html#writing-hook-functions

Plugin Boilerplate

https://github.com/pytest-dev/cookiecutter-pytest-plugin

Installable Plugin

Need to implement `pytest11` entrypoint in `setup.py`, so pytest finds it.

Installable Plugin

```
entry_points={
    'pytest11': [
        'pytest_cov = pytest_cov.plugin',
    ],
    'console_scripts': [
    ]
},
```

https://github.com/pytest-dev/pytest-cov/blob/master/setup.py

Installable Plugin

```
def pytest_addoption(parser):
    # Register argparse and INI options

@pytest.mark.tryfirst
def pytest_load_initial_conftests(early_config, parser, args):
    # Bootstrap setuptools plugin


def pytest_configure(config):
    # Perform initial configuration
```

https://github.com/pytest-dev/pytest-cov/blob/master/src/pytest_cov/plugin.py

Adding Commandline Options

In `conftest.py`:

```
def pytest_addoption(parser):
    parser.addoption('--mac', action='store_true',
                     help='Run Mac tests')
```

In tests:

```
@pytest.fixture
def a_fixture(request):
    mac = request.config.getoption('mac')

def test_foo(pytestconfig):
    mac = pytestconfig.getoption('mac')
```

Assignment

Assignment 8

METASNAKE

3rd Party Plugins

List

Python 2 & 3 compatibility

http://plugincompat.herokuapp.com/

pytest-xdist

Distribute tests among (7) CPUs

```
$ pip install pytest-xdist
$ py.test -n 7
```

pytest-flake8

Run flake8 on all py files

```
$ pip install pytest-flake8
$ py.test --flake8
```

pytest-cov

Run coverage

```
$ pip install pytest-cov
$ py.test --cov=adder --cov-report=html
tests/
# look at htmlcov/index.html
```

Assignment

Assignment 9

METASNAKE

Tox

Tox

3rd party tool for running tests on different pythons

Install

```
pip install tox
```

Configuration

```
# tox.ini
[tox]
envlist = py27,py36
[testenv]
deps=pytest  # use pytest
commands=pytest
```

Configuration

Run `tox-quickstart` to generate config for you

Running

At this point, if you run `tox`, it will:

- Create Python 2.7 venv
  - Install pytest
  - Create sdist and install package
  - Run package tests with pytest
- Create Python 3.6 venv
  - Install pytest
  - Create sdist and install package
  - Run package tests with pytest

Jenkins CI

Can integrate with Jenkins by having Tox installed and having pytest output JunitXML files (with the `--junitxml` option)

CircleCI

Contents of `circle.yml`:

```
dependencies:
    pre:
      - pip install tox

test:
    override:
      - tox
```

Travis CI

Contents of `.travis.yml`:

```
language: python
python:
    - "2.7"
    - "3.6"
install: pip install tox-travis
script: tox
```

Thanks

Go forth and test!