Mastering Pytest

@__mharrison__

METASNAKE

Assignment 1

- Install pytest in a virtual environment
- Run:

```
pytest -h
```

Assignment 2

- Checkout skilift from github https://github.com/mattharrison/skilift
- Create a **`test/`** directory in the checkout
- Create a **`test/test_skilift.py`** file
- Create a test function **`test_line_take`** that creates a **`Line`** with 5 people and takes 7. Ensure that the amount returned is 5 and the **`.num_people`** attribute is 0.
- Create a test function **`test_lift_one_bench`** that creates a line of 5, and a quad lift with 10 benches. Call **`.one_bench`** and assert that the correct results are returned.

Test Parameterization

METASNAKE

Test Parameterization

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

Test Parameterization

Note that the Node Ids change:

```
$ python -m 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 =========
```

METASNAKE

Assignment 3

- Create a test function `test_line_bad`, that creates lines with `[]`, `None`, and `'10'` in them. It tries to call `.take(1)` on each, and checks that the appropriate error is raised.
- Create a test function `test_line_sizes` that creates lines of size 0, 5, and 10. It takes 5, 2, and 0 from each respectively and asserts that the `.num_people` attribute is correct. (You should probably parameterize the result as well)
- Run only the `test_line_sizes` test when the line is created with size 10. (Hint use `-v` to get an id)

Fixtures

Fixtures

Provides consistent tests by dependency injection and setup/teardown

Fixtures

```python
@pytest.fixture
def large_num():
    # assume painful to create
    return 1e20

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

Method Fixtures

```python
class TestAdder:

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

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

Fixtures Parameterization

This will run 3 tests!

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

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

METASNAKE

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 -----
num
    tests/test_funcs_pytest.py:17: no docstring available
```

METASNAKE

Assignment 4

- Create a fixture for a line of size 5. Use that fixture in `test_line_take` and `test_lift_one_bench`
- Create a fixture for a Quad lift of size 10. Use that in `test_lift_one_bench`

More Fixtures

METASNAKE

Teardown in Fixtures

3 ways to insert logic before/after in tests:

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

Module Level

Called once before and after all the functions in the module are called:

```python
def setup_module():
    ...

def teardown_module():
    ...
```

Class Level

Called once for each class:

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

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

METASNAKE

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`

Special fixture. Attributes of the request object:

- `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

```python
@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

```python
@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

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

Fixture Scope

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

METASNAKE

Fixture Scope

```python
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

```python
# 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")
```

METASNAKE

Pass data from marks to fixtures

For pytest >= 3.10 use `.get_closest_marker`

```
@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

```python
@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')
```

METASNAKE

Assignment 5

- Create a fixture, **`line_n`**, that depends on **`request`**. Read off of the marker to get a line size. Create a test, **`test_line_6`**, that creates a tests **`.take`** on a **`Line`** with length 6.
- Create a fixture, **`BenchN`**, that depends on **`request`**. Read off of the marker to get a bench size. Dynamically subclass **`_Bench`** to create a subclass with the passed in size. Create a test, **`test_bench6`**, like **`test_lift_one_bench`**, that uses the fixture to create 6 person bench and test it.

Monkey Patch Fixture

METASNAKE

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

```python
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
```

METASNAKE

Assignment 6

- Create a test, **`test_half_take`**, that monkey patches **`Line.take`** so that only half the amount requested are returned from the line. (ie. **`line.take(4)`** would only take 2 from the line)

METASNAKE

Configuration

Configuration

- Rootdir
  - Node ids determined from root
  - Plugins may store data there
  - Default is where `pytest` is executed
- `pytest.ini` (or `tox.ini` or `setup.cfg`)
  - Must have `[pytest]` section
  - Determines rootdir location (if used)

Hint

Create a `pytest.ini` file (empty is fine) for consistent rootdir

Some INI Options

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

```
$ pytest --help
```

METASNAKE

Some INI Options

- `minversion = 4.0` - Fail if pytest < 4.0
- `addopts = -v` - Add verbose flag (can be overridden by cmd line)
- `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

Example

```
[pytest]
addopts = --doctest-modules -v

markers =
  bad: bad numbers
  large: large numbers
```

METASNAKE

Conftest

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

Assignment 7

- Create a `pytest.ini` file
- Add an option to run the doctests
- Register the missing marks
- Create a `test/conftest.py` file. Move the fixtures to this file

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

Removes tedious package creation:

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

Installable Plugin

pytest looks for **`pytest11`** entrypoint in **`setup.py`**

Installable Plugin

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

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

METASNAKE

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

METASNAKE

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 8

- Examine the **`setup.py`** for **`pytest-cov`** on GitHub.
- What is the entry point?
- What hook does the plugin implement?

3rd Party Plugins

List

Python 2 & 3 compatibility

http://plugincompat.herokuapp.com/

pytest-xdist

Distribute tests among (7) CPUs

```
$ pip install pytest-xdist
$ pytest -n 7
```

METASNAKE

pytest-flake8

Run flake8 on all py files

```
$ pip install pytest-flake8
$ pytest --flake8
```

pytest-cov

Run coverage

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

pytest-faulthandler

Catch segfaults (good for C/C++)

pytest-django

Database access, user/admin fixtures, server fixture

pytest-asyncio

Decorator to mark **`async def`** tests

docker-services

Create (Docker) services that your test need

pytest-selenium

Fixture for automating web applications

pytest-timeout

Mark for timing out after some period

pytest-annotate

Generate type annotations for Pyannotate

pytest-mypy

Run type checks

METASNAKE

Assignment 9

- Install pytest-cov
- Run coverage on the project using the plugin

Thanks!

Go forth and test!

Let's connect on:

- Twitter `@__mharrison__`
- LinkedIn