Introduction to pytest

Why test?

- Testing helps keep code working
  - When developing
  - When maintaining
  - When modifying/enhancing
- Testing gives you confidence in the code now and in the future

What is pytest?

- A test runner – collects and executes test code
- A test framework
  - Rules for test naming
  - Rules for test specification and structure
  - Controls for what tests execute, when and how
  - Display of results, both success and failure

Is it pytest or py.test?

- Originally was part of the 'py' package and was named 'py.test'
- Became its own package and is now called 'pytest'

Alternatives

- DocTest – embedded tests
- UnitTest
  - In the standard library
  - Derivative of the Java Junit library
  - Class oriented
- Nose/Nose2
  - Nose recommends using Nose2
  - Cleaner and easier than UnitTest
  - Originally a fork of pytest

Why pytest?

- Low boilerplate – no mandatory classes
- Use ordinary Assert statements not special functions
- Flexible
  - Runs Nose and UnitTest tests as well
  - Extendable via plugins (~150 available)
    - e.g. Coverage, HTML output, Django, BDD, xDist
- Good error messages
- Well maintained

pytest Basics

- Use pip to install: **pip install pytest**
- Simplest invocation: **pytest**
  - Searches current directory and walks down directories collecting and executing test files
  - Test file naming: ***_test.py** or **test_*.py**
  - Within module: any function or method prefixed with **test_** or class with **Test**
  - Tests may be in the code module or separate

Basic Test Example

Special_math.py

```
def multiply(a, b):
    return a * b
```

Basic_test.py

```
import pytest
from special_math import multiply

def test_multiply():
    assert multiply(2,3) == 6
```

Test Result

```
C:\Users\Dennis\Documents\hgProjects\Pytest Presentation\s08>pytest3
============================= test session starts =============================
platform win32 -- Python 3.5.3, pytest-3.0.7, py-1.4.33, pluggy-0.4.0
rootdir: C:\Users\Dennis\Documents\hgProjects\Pytest Presentation\s08, inifile:
plugins: html-1.13.0, cov-2.4.0
collected 1 items

basic_test.py .

========================== 1 passed in 0.03 seconds ===========================
```

Multiple Similar Tests

- Hard coding is inefficient
- pytest uses a decorator to automate multiple runs of the same test:
  - @pytest.mark.parametrize('variable names', iterable)

Parametrize multiple tests

```
import pytest
from special_math import multiply

multiplications = ((2, 3, 6),
                   (0, 4, 0),
                   (-4, 5, -20),
                   (6, -7, -42),
                   (-8, -9, 72),
                   (3, 'x', 'xxx')
                  )

@pytest.mark.parametrize('a, b, result', multiplications)
def test_multiply(a, b, result):
    assert multiply(a, b) == result
```

pytest .....X.....F.....

Results in 6 tests run

```
C:\Users\Dennis\Documents\hgProjects\Pytest Presentation\s11>pytest3
============================= test session starts =============================
platform win32 -- Python 3.5.3, pytest-3.0.7, py-1.4.33, pluggy-0.4.0
rootdir: C:\Users\Dennis\Documents\hgProjects\Pytest Presentation\s11, inifile:
plugins: html-1.13.0, cov-2.4.0
collected 6 items

parametrize_s11_test.py ......

========================== 6 passed in 0.08 seconds ===========================
```

Using Verbose option

```
C:\Users\Dennis\Documents\hgProjects\Pytest Presentation\s11>pytest3 -v
============================= test session starts =============================
platform win32 -- Python 3.5.3, pytest-3.0.7, py-1.4.33, pluggy-0.4.0 -- c:\user
s\dennis\appdata\local\programs\python\python35-32\python.exe
cachedir: .cache
rootdir: C:\Users\Dennis\Documents\hgProjects\Pytest Presentation\s11, inifile:
plugins: html-1.13.0, cov-2.4.0
collected 6 items

parametrize_s11_test.py::test_multiply[2-3-6] PASSED
parametrize_s11_test.py::test_multiply[0-4-0] PASSED
parametrize_s11_test.py::test_multiply[-4-5--20] PASSED
parametrize_s11_test.py::test_multiply[6--7--42] PASSED
parametrize_s11_test.py::test_multiply[-8--9-72] PASSED
parametrize_s11_test.py::test_multiply[3-x-xxx] PASSED

========================== 6 passed in 0.12 seconds ===========================
```

Add a new feature

- Let's say we want to expand the multiply function to allow multiplication by None
- Should give 0 as a result
- Add a new test condition to the multiplication tests
  - (None, 5, 0)

Expanded test

```
import pytest
from special_math import multiply

multiplications = ((2, 3, 6),
                   (0, 4, 0),
                   (-4, 5, -20),
                   (6, -7, -42),
                   (-8, -9, 72),
                   (3, 'x', 'xxx'),
                   (None, 5, 0)
                  )

@pytest.mark.parametrize('a, b, result', multiplications)
def test_multiply(a, b, result):
    assert multiply(a, b) == result
```

pytest .....X.....F.....

Run New Tests

```
============================= test session starts =============================
platform win32 -- Python 3.5.3, pytest-3.0.7, py-1.4.33, pluggy-0.4.0
rootdir: C:\Users\Dennis\Documents\hgProjects\Pytest Presentation\s15, inifile:
plugins: html-1.13.0, cov-2.4.0
collected 7 items

parametrize_s15_test.py ......F

================================== FAILURES ===================================
_____________________________ test_multiply[None-5-0] _____________________________

a = None, b = 5, result = 0

    @pytest.mark.parametrize('a, b, result', multiplications)
    def test_multiply(a, b, result):
>       assert multiply(a, b) == result

parametrize_s15_test.py:15:
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

a = None, b = 5

    def multiply(a, b):
>       return a * b
E       TypeError: unsupported operand type(s) for *: 'NoneType' and 'int'

special_math.py:3: TypeError
====================== 1 failed, 6 passed in 0.22 seconds ======================
```

pytest .....X.....F.....

Fix the multiply function

```
def multiply(a, b):
    if a is None or b is None:
        return 0
    return a * b
```

and rerun tests

```
============================= test session starts =============================
platform win32 -- Python 3.5.3, pytest-3.0.7, py-1.4.33, pluggy-0.4.0
rootdir: C:\Users\Dennis\Documents\hgProjects\Pytest Presentation\s15, inifile:
plugins: html-1.13.0, cov-2.4.0
collected 7 items

parametrize_s15_test.py .......

========================== 7 passed in 0.08 seconds ===========================
```

Other “mark” Decorators

- Builtins
  - xfail
  - skip, skipif
  - tryfirst, trylast
  - slowtest
- Custom data (can be used to identify which tests to run/not run)
  - e.g. @pytest.mark.webtest
  - Then pytest -m webtest or pytest -m “not webtest”

Fixtures: Setup/Tear-down and more

- Pytest uses a “fixture” decorator
  - extends simple setup/teardown methods
  - Named: test functions specify which fixtures to use
- @pytest.fixture
- Simple example:

```
@pytest.fixture
def initial_gui():
    app = wall_builder_gui.WallBuilderApp()
    app.gui.Show(False)
    return app.gui
```

Using a fixture

Fixture:

```
@pytest.fixture
def initial_gui():
    app = wall_builder_gui.WallBuilderApp()
    app.gui.Show(False)
    return app.gui
```

Use:

```
handlers = ['on_preview', 'on_create']

@pytest.mark.parametrize("handler", handlers)
def test_event_handlers_exist(initial_gui, handler):
    assert getattr(initial_gui, handler) is not None
```

Test Run

```
C:\Users\Dennis\Documents\hgProjects\Pytest Presentation\s20>pytest3 -v
============================= test session starts =============================
platform win32 -- Python 3.5.3, pytest-3.0.7, py-1.4.33, pluggy-0.4.0 -- c:\user
s\dennis\appdata\local\programs\python\python35-32\python.exe
cachedir: .cache
rootdir: C:\Users\Dennis\Documents\hgProjects\Pytest Presentation\s20, inifile:
plugins: html-1.13.0, cov-2.4.0
collected 2 items

fixture_s20_test.py::test_event_handlers_exist[on_preview] PASSED
fixture_s20_test.py::test_event_handlers_exist[on_create] PASSED

========================== 2 passed in 0.41 seconds ===========================
```

Fixture with options

Scoping, Naming, Teardown:

```
@pytest.fixture(scope='function', name='initial_gui')
def fixture_initial_gui():
    app = wall_builder_gui.WallBuilderApp()
    app.gui.Show(False)
    yield app.gui

    app.Destroy()
```

Fixture Parametrization

- Use **'params'** keyword argument to pass in an iterable
- Fixture run once in turn for each item in the iterable
- Can be accessed in fixture function via special **'request'** object via **'request.param'**

Parametrized Fixture Example

Say we have the following duck.py module:

```
class Duck:
    def quack(self, count):
        return ', '.join(count*['quack'])

    def swim(self):
        return 'swimming'

class RoboDuck(Duck):
    def fire_lasers(self):
        return 'lasers firing'
```

Parametrized Fixture Example

```
import pytest
from duck import Duck, RoboDuck

@pytest.fixture(params=[Duck, RoboDuck], name='ducks')
def fixture_ducks(request):
    return request.param()

quacks = ((0, ''), (1, 'quack'), (2, 'quack, quack'))

@pytest.mark.parametrize('repeats, result', quacks)
def test_duck_talk(ducks, repeats, result):
    assert ducks.quack(repeats) == result

def test_duck_swim(ducks):
    assert ducks.swim() == 'swimming'
```

pytest .....X.....F.....

Test Result with -v option

```
============================= test session starts =============================
platform win32 -- Python 3.5.3, pytest-3.0.7, py-1.4.33, pluggy-0.4.0 -- c:\user
s\dennis\appdata\local\programs\python\python35-32\python.exe
cachedir: .cache
rootdir: C:\Users\Dennis\Documents\hgProjects\Pytest Presentation\s25, inifile:
plugins: html-1.13.0, cov-2.4.0
collected 8 items

fixture_s25_test.py::test_duck_talk[Duck-0-] PASSED
fixture_s25_test.py::test_duck_talk[Duck-1-quack] PASSED
fixture_s25_test.py::test_duck_talk[Duck-2-quack, quack] PASSED
fixture_s25_test.py::test_duck_talk[RoboDuck-0-] PASSED
fixture_s25_test.py::test_duck_talk[RoboDuck-1-quack] PASSED
fixture_s25_test.py::test_duck_talk[RoboDuck-2-quack, quack] PASSED
fixture_s25_test.py::test_duck_swim[Duck] PASSED
fixture_s25_test.py::test_duck_swim[RoboDuck] PASSED

========================== 8 passed in 0.11 seconds ===========================
```

Some Additional Features

- Running:
  - Selective tests, recursion discovery control
  - Naming convention control
  - Compatible with CI systems like Jenkins
- Configuration files:
  - pytest.ini and conftest.py
  - Can set options and contain common Fixtures
- Parametrize:
  - Can be 'stacked', all combinations will be run
- Fixtures:
  - Can have 'ids' to specially identify tests
  - Fixtures can use other Fixtures

Resources

- Web site/docs: https://docs.pytest.org/en/latest/
- PYPI: https://pypi.python.org/pypi/pytest/3.1.2
- Tutorials: http://pythontesting.net/framework/pytest/pytest-introduction/

Extra Credit

- Tests for special_math.py Multiplication missed a major type of test – floats
- Add a new test
  - (0.3, 3, 0.9)
  - Running test fails
- Fix test function

New Test File

```
import pytest
from special_math import multiply

multiplications = ((2, 3, 6),
                   (0, 4, 0),
                   (-4, 5, -20),
                   (6, -7, -42),
                   (-8, -9, 72),
                   (3, 'x', 'xxx'),
                   (None, 5, 0),
                   (0.3, 3, 0.9)
                   )

@pytest.mark.parametrize('a, b, result', multiplications)
def test_multiply(a, b, result):
    assert multiply(a, b) == result
```

pytest .....X.....F.....

Failed test

```
collected 8 items

parametrize_s29_test.py::test_multiply[2-3-6] PASSED
parametrize_s29_test.py::test_multiply[0-4-0] PASSED
parametrize_s29_test.py::test_multiply[-4-5--20] PASSED
parametrize_s29_test.py::test_multiply[6--7--42] PASSED
parametrize_s29_test.py::test_multiply[-8--9-72] PASSED
parametrize_s29_test.py::test_multiply[3-x-xxx] PASSED
parametrize_s29_test.py::test_multiply[None-5-0] PASSED
parametrize_s29_test.py::test_multiply[0.3-3-0.9] FAILED

================================== FAILURES ===================================
_____________________________ test_multiply[0.3-3-0.9] _____________________________

a = 0.3, b = 3, result = 0.9

    @pytest.mark.parametrize('a, b, result', multiplications)
    def test_multiply(a, b, result):
>       assert multiply(a, b) == result
E       assert 0.8999999999999999 == 0.9
E        +  where 0.8999999999999999 = multiply(0.3, 3)

parametrize_s29_test.py:16: AssertionError
==================== 1 failed, 7 passed in 0.19 seconds =====================
```

Fixed test file

```
import pytest
import math
from special_math import multiply

multiplications = ((2, 3, 6),
                   (0, 4, 0),
                   (-4, 5, -20),
                   (6, -7, -42),
                   (-8, -9, 72),
                   (3, 'x', 'xxx'),
                   (None, 5, 0),
                   (0.3, 3, 0.9)
                   )

@pytest.mark.parametrize('a, b, result', multiplications)
def test_multiply(a, b, result):
    if isinstance(a, float) or isinstance(b, float):
        assert math.isclose(multiply(a, b), result)
    else:
        assert multiply(a, b) == result
```

pytest .....X.....F.....

Fixed

```
collected 8 items

parametrize_s31_test.py::test_multiply[2-3-6] PASSED
parametrize_s31_test.py::test_multiply[0-4-0] PASSED
parametrize_s31_test.py::test_multiply[-4-5--20] PASSED
parametrize_s31_test.py::test_multiply[6--7--42] PASSED
parametrize_s31_test.py::test_multiply[-8--9-72] PASSED
parametrize_s31_test.py::test_multiply[3-x-xxx] PASSED
parametrize_s31_test.py::test_multiply[None-5-0] PASSED
parametrize_s31_test.py::test_multiply[0.3-3-0.9] PASSED

========================== 8 passed in 0.09 seconds ==========================
```