Unit testing

•••

Kim Brugger

When a program is being tested,
it is too late to make design changes

-The Tao of Programming

Unit testing: Outline

Introduction

Pytest framework

Hands on

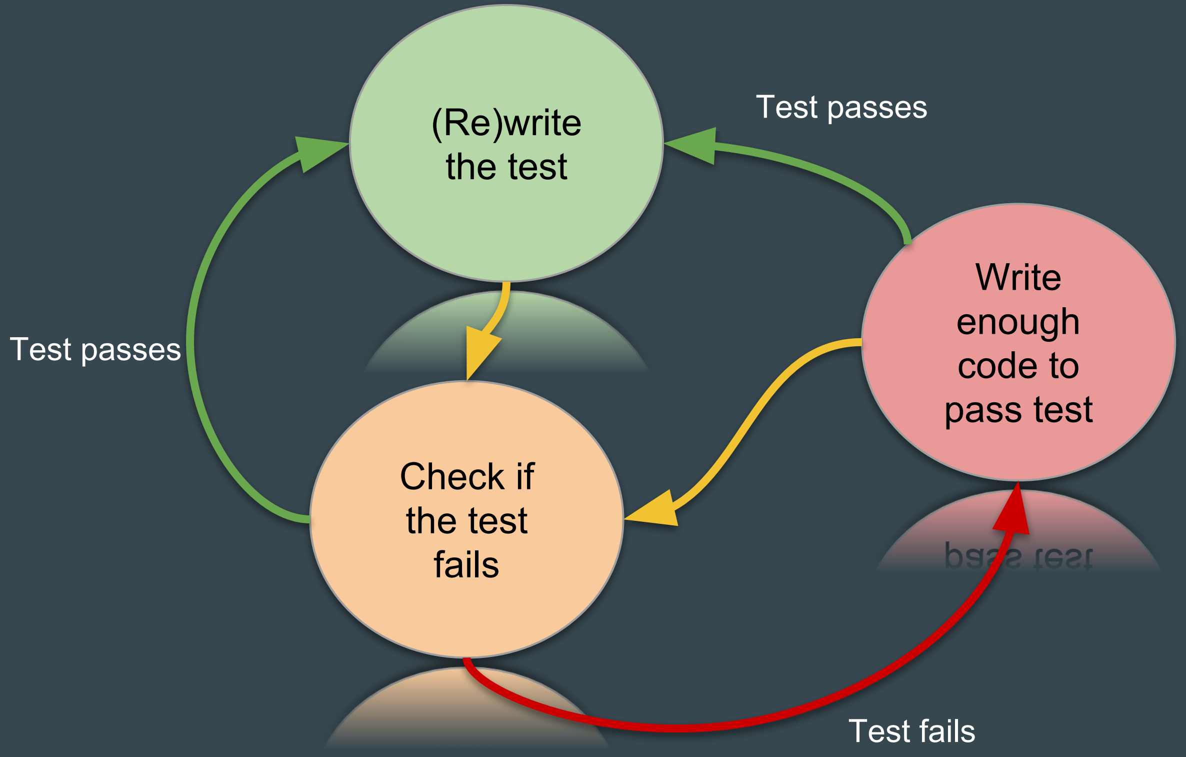
Reflection

Introduction: Unit testing

Write unit tests to improve productivity as a programmer.

Unit tests are highly localized. Each test works within a single package. It tests the interfaces, but beyond that it assumes the rest just works

Introduction: Test driven development



Introduction: benefits

1. Makes the development process flexible
2. Help the overall design
3. Improve overall quality of code
4. Identify bugs early
5. Facilitates changes and
6. Simplifies integration

Pytest: Introduction

```python
# content of test_sample.py
def inc(x):
    return x + 1


def test_answer():
    assert inc(3) == 5



if __name__ == '__main__':

      main()
```

Pytest: result of doing a test

```
pytest
=========================== test session starts ============================
platform linux -- Python 3.x.y, pytest-4.x.y, py-1.x.y, pluggy-0.x.y
cachedir: $PYTHON_PREFIX/.pytest_cache
rootdir: $REGENDOC_TMPDIR, inifile:
collected 1 item

test_sample.py F                                                     [100%]

================================= FAILURES =================================
_______________________________ test_answer ________________________________

    def test_answer():
>       assert inc(3) == 5
E       assert 4 == 5
E        +  where 4 = inc(3)

test_sample.py:6: AssertionError
========================= 1 failed in 0.12 seconds =========================
```

Pytest: Advanced Testing

```
# Catch an exception
import pytest
def test_exception():
  with pytest.raises( TypeError):
      raise os.path.isfile(‘not a file’)

# capture printed content ( pytest --capture=sys)
def test_myoutput(capsys):
   captured = capsys.readouterr()
   assert captured.out == "hello\n"
   assert captured.err == "world\n"

# emulate output from other packages/libraries
from unittest.mock import Mock, patch

@patch.object(htcondor.Schedd, 'xquery', fake_job_query)
def test_job_counts():
   job_counts = c.job_counts()
```

Pytest: Unit tests are hard to get good

```
def q_construct (sql:str, limit:str, order:str):

    if limit is not None;
        sql += " limit {}".format( limit )

    if order is not None;
        sql += " order by {}".format( order )

    return db.do( sql )
```

Pytest: Single function example

```
def add_one(data):

    if ( not isinstance(data, list)):
        raise TypeError

    new_list = [ ]
    for d in data:
         new_list.append( d + 1 )

    return new_list
```

```
def test_one_single():
    assert add_one([1]) == [2]

def test_one_multi():
    assert add_one([1,2]) == [2,3]

def test_one_str():
   with pytest.raises( TypeError):
     add_one(1,2)

def test_one_empty():
    assert add_one([ ]) == [ ]
```

Pytest: Single function example II

```
pytest -v ./src/unit_test_example.py
====================================== test session starts =======================================
platform linux -- Python 3.5.3, pytest-4.2.0, py-1.7.0, pluggy-0.8.1 --
...
collected 4 items

src/unit_test_example.py::test_one_singlePASSED                                                   [ 25%]
src/unit_test_example.py::test_one_multiPASSED                                                    [ 50%]
src/unit_test_example.py::test_one_strPASSED                                                      [ 75%]
src/unit_test_example.py::test_one_emptyPASSED                                                    [100%]

==================================== 4 passed in 0.01 seconds ====================================
```

pytest commands

```
#run tests in a single test file
pytest-3 -x t/project/module_test.py

#halt after first failed test
pytest-3 -x t/project/module_test.py

#verbose output on fails
pytest-3 -v t/project/module_test.py

#run test names matching a pattern
pytest-3 -k 'connect' t/project/module_test.py

#run test names not matching a pattern
pytest-3 -k 'not connect' t/project/module_test.py
```

```
#run test names matching one or more patterns (or not)

pytest-3 -k 'cloud and connect' t/project/module_test.py
pytest-3 -k 'cloud and not unknown' t/project/module_test.py
pytest-3 -k 'cloud or server' t/project/module_test.py


#show percentage of code covered (and lines missed)
pytest-3 --cov=project.module t/project/
pytest-3 --cov-report term-missing --cov=project t/project/
```

Hands on (45 min)

Write unit tests for word_counter.py

What was your strategy?

What problems did you face?

Which are easier to test and why?

Anything you would change in the program flow?

Unit testing: reflection & thoughts

How can unit testing help in your project?

How hard will it be to add to an existing project?

What is the overhead of doing unit testing?

It it possible to cover all cases