



Kyoto, Japan in 2010
by Usa Sammapun

06 Unit Testing

Usa Sammapun

Outline

- พื้นฐานของการทดสอบ
- การทดสอบระดับหน่วย (unit testing) คืออะไร
 - เครื่องมือการทดสอบ unittest library
- Fixtures

assert

- บอก python ว่า ก่อนจะทำงานต่อ โปรแกรมจะต้องตรงกับเงื่อนไขใด
- โปรแกรมจะหยุดทำงาน เมื่อเจอ assert อันแรก fail

```
def divide(a, b):  
    assert b != 0  
    # ..... code here .....  
  
print "output =", divide(2,2)  
print "output =", divide(2,0)
```

```
$ python calculator.py  
output = 1  
output =  
Traceback (most recent call last):  
  File "calculator.py", line 20, in <module>  
    print "output =", divide(2,0)  
  File "calculator.py", line 16, in divide  
    assert b != 0  
AssertionError  
$
```

unittest library และการใช้งานเบื้องต้น

unittest library

- มากับ standard library ของ python
- จัดโมดูลหรือไฟล์ให้เหมาะสม

lab02.py

```
def isprime(n):  
    # .... your code here ....  
  
def iseven(n):  
    # .... your code here ....
```

ตั้งชื่อให้เหมาะสม เช่น
test_ชื่อโมดูล_ชื่อฟังก์ชัน.py

test_lab02_iseven.py

```
import unittest  
import lab02  
  
class TestIsEven(unittest.TestCase):  
    def test_even(self):  
        assert lab02.iseven(2)  
    def test_odd(self):  
        assert not lab02.iseven(3)  
  
if __name__ == '__main__':  
    unittest.main()
```


เขียน unit test ด้วย unittest library (1)

test_lab02_iseven.py

```
import unittest
import lab02

class TestIsEven(unittest.TestCase):
    def test_even(self):
        assert lab02.iseven(2)
    def test_odd(self):
        assert not lab02.iseven(3)

if __name__ == '__main__':
    unittest.main()
```

import unittest library ของ python

เขียน unit test ด้วย unittest library (2)

test_lab02_iseven.py

```
import unittest
import lab02

class TestIsEven(unittest.TestCase):
    def test_even(self):
        assert lab02.iseven(2)
    def test_odd(self):
        assert not lab02.iseven(3)

if __name__ == '__main__':
    unittest.main()
```

import โมดูลที่ต้องการจะทดสอบ

เขียน unit test ด้วย unittest library (3)

test_lab02_iseven.py

```
import unittest
import lab02

class TestIsEven(unittest.TestCase):
    def test_even(self):
        assert lab02.iseven(2)
    def test_odd(self):
        assert not lab02.iseven(3)

if __name__ == '__main__':
    unittest.main()
```

จับกลุ่มกรณีทดสอบไว้
ด้วยกันใน “คลาส”

เขียน unit test ด้วย unittest library (4)

test_lab02_iseven.py

```
import unittest
import lab02

class TestIsEven(unittest.TestCase):
    def test_even(self):
        assert lab02.iseven(2)
    def test_odd(self):
        assert not lab02.iseven(3)

if __name__ == '__main__':
    unittest.main()
```

สร้างกรณีทดสอบ โดย 1 เทสคือ 1 ฟังก์ชัน

ชื่อฟังก์ชัน ต้องขึ้นต้น
ด้วย test_

เขียน unit test ด้วย unittest library (5)

test_lab02_iseven.py

```
import unittest
import lab02

class TestIsEven(unittest.TestCase):
    def test_even(self):
        assert lab02.iseven(2)
    def test_odd(self):
        assert not lab02.iseven(3)

if __name__ == '__main__':
    unittest.main()
```

ใช้ assert ในการตรวจสอบว่า
ผลที่คาด == ผลลัพธ์จริง หรือไม่
expected result == actual result ??

เขียน unit test ด้วย unittest library (6)

test_lab02_iseven.py

```
import unittest
import lab02

class TestIsEven(unittest.TestCase):
    def test_even(self):
        assert lab02.iseven(2)
    def test_odd(self):
        assert not lab02.iseven(3)

if __name__ == '__main__':
    unittest.main()
```

ควรคิด กรณีทดสอบ ให้ครบถ้วน

เขียน unit test ด้วย unittest library (7)

test_lab02_iseven.py

```
import unittest
import lab02

class TestIsEven(unittest.TestCase):
    def test_even(self):
        assert lab02.iseven(2)
    def test_odd(self):
        assert not lab02.iseven(3)

if __name__ == '__main__':
    unittest.main()
```

บอกให้ python รันเทสของเรา

เขียนเสร็จแล้ว เรามารันกัน (1)

- ผ่าน !

```
def iseven(n):  
    result = False  
    if (n%2 == 0):  
        result = True  
    return result
```

```
$ python test_lab02_iseven.py
```

```
..
```

```
-----  
Ran 2 tests in 0.000s
```

```
OK
```

```
$
```

2 จุดนี้บอกว่า รัน 2 เทสและผ่านทั้งคู่

เขียนเสร็จแล้ว เรามารันกัน (2)

- อู๋บส์ เขียนโค้ดผิดไปนิด

```
def iseven(n):  
    result = True  
    if (n%2 == 0):  
        result = True  
    return result
```

```
$ python test_lab02_iseven.py  
.F
```

fail ตรงเทสที่ 2

```
=====
```

```
FAIL: test_odd (__main__.TestIsEven)
```

```
-----
```

```
Traceback (most recent call last):
```

```
  File "test_lab02_iseven.py", line 8, in test_odd  
    assert not lab02.iseven(3)
```

```
AssertionError
```

```
-----
```

```
Ran 2 tests in 0.000s
```

```
FAILED (failures=1)
```

```
$
```

เขียนเสร็จแล้ว เรามารันกัน (3)

- อู๋บส์ เขียน **test** ผิดไปนิด

```
class TestIsEven(unittest.TestCase):  
    def test_even(self):  
        assert lab02.isEven(2)  
    def test_odd(self):  
        assert not lab02.iseven(3)
```

```
$ python test_lab02_iseven.py
```

```
E.
```

error ตรงเทสที่ 1

```
=====
```

```
ERROR: test_even (__main__.TestIsEven)
```

```
-----
```

```
Traceback (most recent call last):
```

```
  File "test_lab02_iseven.py", line 6, in test_even
```

```
    assert lab02.isEven(2)
```

```
AttributeError: 'module' object has no attribute  
'isEven'
```

```
-----
```

```
Ran 2 tests in 0.000s
```

```
FAILED (errors=1)
```

```
$
```


การเขียน assert ด้วยฟังก์ชันใน unittest.TestCase

- self.assertTrue() และ self.assertFalse()

assert ธรรมดา

```
import unittest
import lab02

class TestIsEven(unittest.TestCase):
    def test_even(self):
        assert lab02.iseven(2)
    def test_odd(self):
        assert not lab02.iseven(3)

if __name__ == '__main__':
    unittest.main()
```

ใช้ฟังก์ชันของ TestCase

```
import unittest
import lab02

class TestIsEven(unittest.TestCase):
    def test_even(self):
        self.assertTrue(lab02.iseven(2))
    def test_odd(self):
        self.assertFalse(lab02.iseven(3))

if __name__ == '__main__':
    unittest.main()
```

การเขียน assert ด้วยฟังก์ชันใน unittest.TestCase

- self.assertEqual(expected result, actual result)

assert ธรรมดา

```
import unittest
import calc

class TestAdd(unittest.TestCase):
    def test_add_onedigit(self):
        assert calc.add(2,3) == 5
    def test_add_twodigit(self):
        assert calc.add(12,25) == 37

if __name__ == '__main__':
    unittest.main()
```

ใช้ฟังก์ชันของ TestCase

```
import unittest
import calc

class TestAdd(unittest.TestCase):
    def test_add_onedigit(self):
        self.assertEqual(5, calc.add(2,3))
    def test_add_twodigit(self):
        self.assertEqual(37, calc.add(12,25))

if __name__ == '__main__':
    unittest.main()
```

การเขียน assert ด้วยฟังก์ชันใน unittest.TestCase

- เก็บค่าไว้ ก่อนใช้ assert
 - แยกการรัน CUT (code under test) กับการตรวจสอบออกจากกัน

```
import unittest
import calc

class TestAdd(unittest.TestCase):
    def test_add_onedigit(self):
        result = calc.add(2,3)
        self.assertEqual(5, result)
    def test_add_twodigit(self):
        result = calc.add(12,25)
        self.assertEqual(37, result)

if __name__ == '__main__':
    unittest.main()
```

รันเทสด้วย -v

- แสดงรายละเอียดเพิ่มขึ้น

```
$ python test_calc_add.py
```

```
..
```

```
-----  
Ran 2 tests in 0.000s
```

```
OK
```

```
$
```

```
$ python test_calc_add.py -v
```

```
test_add_onedigit (__main__.TestAdd) ... ok
```

```
test_add_twodigit (__main__.TestAdd) ... ok
```

```
-----  
Ran 2 tests in 0.000s
```

```
OK
```

```
$
```

ทดสอบหลายค่า ด้วย action เดียวกัน (1)

```
import unittest
import roman1

class TestRoman(unittest.TestCase):

    known_values = ( (1, 'I'),
                     (2, 'II'),
                     (4, 'IV'),
                     (5, 'V')
                     .....
                   )

    def test_to_roman_known_values(self):

        '''to_roman should give known result with known input'''

        for integer, numeral in self.known_values:
            result = roman1.to_roman(integer)
            self.assertEqual(numeral, result)

if __name__ == '__main__':
    unittest.main()
```

เก็บ input และ expected result

ทดสอบหลายค่า ด้วย action เดียวกัน (2)

```
import unittest
import roman1

class TestRoman(unittest.TestCase):

    known_values = ( (1, 'I'),
                     (2, 'II'),
                     (4, 'IV'),
                     (5, 'V')
                     .....
                   )

    def test_to_roman_known_values(self):

        '''to_roman should give known result with known input'''

        for integer, numeral in self.known_values:
            result = roman1.to_roman(integer)
            self.assertEqual(numeral, result)

if __name__ == '__main__':
    unittest.main()
```

ไม่ต้องใส่ทุกค่า แต่เป็นค่าที่ดูว่าน่า
จะมีความแตกต่างกัน

ทดสอบหลายค่า ด้วย action เดียวกัน (3)

```
import unittest
import roman1

class TestRoman(unittest.TestCase):

    known_values = ( (1, 'I'),
                     (2, 'II'),
                     (4, 'IV'),
                     (5, 'V')
                     .....
                   )

    def test_to_roman_known_values(self):

        '''to_roman should give known result with known input'''

        for integer, numeral in self.known_values:
            result = roman1.to_roman(integer)
            self.assertEqual(numeral, result)

if __name__ == '__main__':
    unittest.main()
```

loop เพื่อเรียกฟังก์ชัน
to_roman และเปรียบเทียบ
ผลลัพธ์ในแต่ละกรณีใน
known_values

ทดสอบหลายค่า ด้วย action เดียวกัน (4)

```
import unittest
import roman1

class TestRoman(unittest.TestCase):

    known_values = ( (1, 'I'),
                     (2, 'II'),
                     (4, 'IV'),
                     (5, 'V')
                     .....
                   )

    def test_to_roman_known_values(self):

        '''to_roman should give known result with known input'''

        for integer, numeral in self.known_values:
            result = roman1.to_roman(integer)
            self.assertEqual(numeral, result)

if __name__ == '__main__':
    unittest.main()
```

เรียกฟังก์ชัน to_roman ให้
ทำงานจริง

ทดสอบหลายค่า ด้วย action เดียวกัน (5)

```
import unittest
import roman1

class TestRoman(unittest.TestCase):

    known_values = ( (1, 'I'),
                     (2, 'II'),
                     (4, 'IV'),
                     (5, 'V')
                     .....
                   )

    def test_to_roman_known_values(self):

        '''to_roman should give known result with known input'''

        for integer, numeral in self.known_values:
            result = roman1.to_roman(integer)
            self.assertEqual(numeral, result)

if __name__ == '__main__':
    unittest.main()
```

เปรียบเทียบผลลัพธ์ใน
แต่ละกรณีใน known_values

ทดสอบหลายค่า ด้วย action เดียวกัน (6)

```
import unittest
import roman1

class TestRoman(unittest.TestCase):


    known_values = ( (1, 'I'),
                     (2, 'II'),
                     (4, 'IV'),
                     (5, 'V')
                     .....
                   )

    def test_to_roman_known_values(self):

        '''to_roman should give known result with known input'''

        for integer, numeral in self.known_values:
            result = roman1.to_roman(integer)
            self.assertEqual(numeral, result)

if __name__ == '__main__':
    unittest.main()
```



แต่การมี loop ใน
เทส ทำให้เทสซับซ้อน มี
bug ง่าย

ทดสอบหลายค่า ด้วย action เดียวกัน (7)

```
import unittest
import roman1

class TestRoman(unittest.TestCase):

    known_values = ( (1, 'I'),
                     (2, 'II'),
                     (4, 'IV'),
                     (5, 'V')
                     .....
                   )

    def test_to_roman_known_values(self):

        '''to_roman should give known result with known input'''

        for integer, numeral in self.known_values:
            result = roman1.to_roman(integer)
            self.assertEqual(numeral, result)

if __name__ == '__main__':
    unittest.main()
```

ทดสอบหยุดหลังจาก
assert แรก fail ทำให้
ทดสอบถัดมาไม่ถูกรัน

ใช้
“parameterized
test” (ไม่อยู่ใน
unittest library)

assertEqual

- ใช้ได้กับหลาย data type
 - เราเห็น integer
- list
 - `self.assertEqual([0,1,2,3,4,5,6,7,8,9], range(10))`
- string
 - `self.assertEqual('dna', 'dna')`
- อื่น เช่น tuple, dict, set เป็นต้น

ใส่คำอธิบายได้ด้วยใน assertEquals

```
def test_unit_with_itself(self):
    unit = ((0,0), (1,1))
    result = overlap(unit, unit)
    self.assertEqual(unit, result,
                     'same rectangle should return itself')
```

```
.F....
=====
FAIL: test_unit_with_itself (__main__.TestCompare)
-----
Traceback (most recent call last):
  File "test_overlap.py", line 16, in test_unit_with_itself
    self.assertEqual(unit, result, 'same rectangle should return
itself')
AssertionError: same rectangle should return itself

-----
Ran 6 tests in 0.004s

FAILED (failures=1)
```


assert อื่นๆ

Method	Checks that	New in
<code>assertEqual(a, b)</code>	<code>a == b</code>	
<code>assertNotEqual(a, b)</code>	<code>a != b</code>	
<code>assertTrue(x)</code>	<code>bool(x)</code> is True	
<code>assertFalse(x)</code>	<code>bool(x)</code> is False	
<code>assertIs(a, b)</code>	<code>a is b</code>	2.7
<code>assertIsNot(a, b)</code>	<code>a is not b</code>	2.7
<code>assertIsNone(x)</code>	<code>x is None</code>	2.7
<code>assertIsNotNone(x)</code>	<code>x is not None</code>	2.7
<code>assertIn(a, b)</code>	<code>a in b</code>	2.7
<code>assertNotIn(a, b)</code>	<code>a not in b</code>	2.7
<code>assertIsInstance(a, b)</code>	<code>isinstance(a, b)</code>	2.7
<code>assertNotIsInstance(a, b)</code>	<code>not isinstance(a, b)</code>	2.7

Fixtures ใน unittest library

Fixtures ใน unittest

```
class TestIsEven(unittest.TestCase):
    def setUp(self):
        print "this is setup"
    def test_even(self):
        assert lab02.iseven(2)
        print "even test"
    def test_odd(self):
        assert not lab02.iseven(3)
        print "odd test"
```



setUp(self) ทำงาน
ก่อนรันแต่ละเทส

```
$ python test_lab02_iseven.py
this is setup
even test
.this is setup
odd test
.
-----
Ran 2 tests in 0.000s

OK
$
```