ch16 文档和测试

vi fkmodule.py

my\_name="crazy software center"

def say\_hi(name):

''' define hello function '''

print("execute say\_hi")

print(name + " hello!")

def print\_rect(hi, wi):

''' define a rectangle print function

hi: height of rectangle

wi: width of rectangle '''

print(('\*'\*wi + '\n') \* hi)

class user:

national = 'China'

''' define a user class

the class includes two variable:

name and age'''

def \_\_init\_\_(self, name, age):

''' constructor function '''

self.name = name

self.age = age

def eat(self, food):

''' define user what to eat '''

print('%s is eating %s' % (self.name, food))

#say\_hi('zcw')

#print\_rect(4,3)

#u = user('wangwu', 17)

#u.eat('apple')

# pydoc

python -m pydoc fkmodule #console

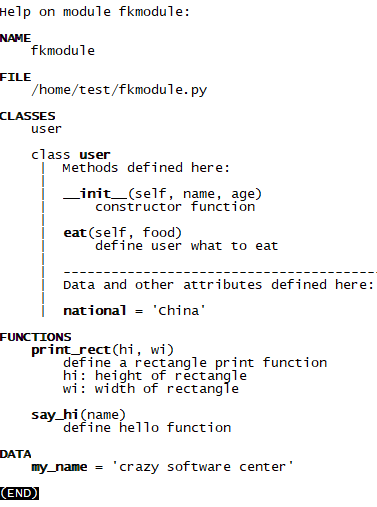
python -m pydoc -w fkmodule #html

## console

#console输出不能持久化保存

#包含模块名，全局函数，全局数据，类，源程序绝对路径

python -m pydoc fkmodule



## 生成html

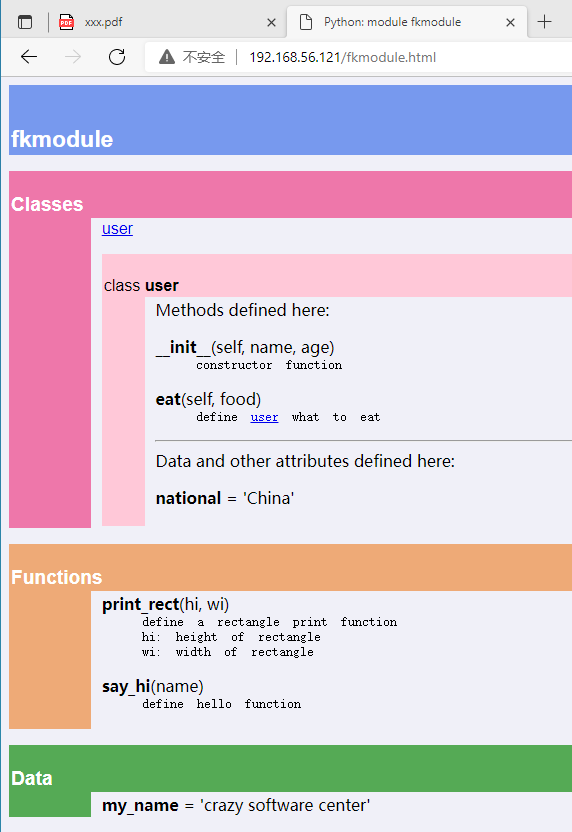
root@2004a:/home/test# python -m pydoc -w fkmodule

wrote fkmodule.html

root@2004a:/home/test# ls

fkmodule.html fkmodule.py fkmodule.pyc

root@2004a:/home/test# cp fkmodule.html /var/www/html/



# 常见的bug管理工具

Bugzilla

BugFree

TestDirector

JIRA

ClearQuest

MantisBT

# 文档测试doctest

import doctest

doctest.testmod()

如果文档中的测试用例都执行正确，则不产生输出

def square(x):

'''

>>> square(2)

4

>>> square(3)

6

'''

return x\*\*2

class user:

'''

define a class

example:

>>> u = user('zcw', 13)

>>> u.name

'zcw'

>>> u.age

13

>>> u.say('i love python')

'zcw say: i love python'

'''

def \_\_init\_\_(self, name, age):

self.name = 'zcw'

self.age = age

def say(self, content):

return self.name + " say: " + content

if \_\_name\_\_ == '\_\_main\_\_':

import doctest

doctest.testmod()

root@2004a:/home/test# python3 test.py

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

File "test.py", line 5, in \_\_main\_\_.square

Failed example:

square(3)

Expected:

6

Got:

9

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1 items had failures:

1 of 2 in \_\_main\_\_.square

\*\*\*Test Failed\*\*\* 1 failures.

root@2004a:/home/test#

# 使用PyUnit(unittest)

## 简介

PyUnit是python自带的测试框架，用于编写和运行可重复的测试，是xUnit体系的一个成员

xUnit是众多测试框架的总称。

PyUnit是xUnit体系的一员，主要用于白盒测试和回归测试。

PyUnit的好处：

产品代码和测试代码分离

针对某类的测试代码只需要较少的改动

PyUnit是开源代码，可以进行二次开发

PyUnit具有如下特征：

使用断言方法判断期望值和实际值的差异，返回bool值

使用测试包，对多个模块批量测试

测试驱动设备可使用共同的初始化变量或实例(setUp,tearDown)

测试驱动开发：开发某个功能前，先定义好该功能的测试结果

## 源程序

def one\_equ(a, b):

''' solve a\*x+b = 0 '''

if a == 0:

raise ValueError('[one\_equ] invalid param')

else:

return -b/a

def two\_equ(a,b,c):

''' slove a\*x\*x+b\*x+c = 0 '''

if a == 0:

raise ValueError('[two\_equ] invalid param')

elif b\*b-4\*a\*c > 0:

r1 = (-b + (b\*b-4\*a\*c)\*\*0.5)/(2\*a)

r2 = (-b - (b\*b-4\*a\*c)\*\*0.5)/(2\*a)

return r1,r2

elif b\*b-4\*a\*c == 0:

return -b/(2\*a)

else:

raise ValueError("no solve")

#r1,r2 = two\_equ(1, -3, 2)

#print(r1,r2)

## 测试桩

root@2004a:/home/test# cat test\_fk\_math.py

import unittest

from fk\_math import \*

class mytest(unittest.TestCase):

def test\_one\_equ(self):

self.assertEqual(one\_equ(5, 9), -1.8)

self.assertTrue(one\_equ(4, 10) == -2.5, .000001)

self.assertTrue(one\_equ(4, -27) == 27/4)

with self.assertRaises(ValueError):

one\_equ(0, 9)

def test\_two\_equ(self):

r1,r2 = two\_equ(1,-3,2)

#elf.assertCountEqual((r1,r2), (1.0,2.0), 'resolve error')

#elf.assertCountEqual((r1,r2), (1.05,2.0), 'resolve error')

self.assertCountEqual((r1,r2), (1.05,2.05), 'resolve error')

r1,r2 = two\_equ(2,-7,6)

self.assertCountEqual((r1,r2), (1.5,2.0), 'resolve error')

r = two\_equ(1,-4,4)

self.assertEqual(r, 2.0, "resolve error")

with self.assertRaises(ValueError):

two\_equ(0, 9, 3)

with self.assertRaises(ValueError):

two\_equ(4, 2, 3)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

## 运行测试

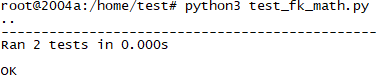
if \_\_name\_\_ == '\_\_main\_\_':

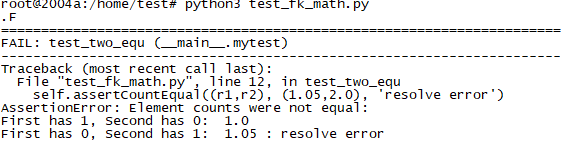
unittest.main() #会自动创建ittest.TestCase的继承类对象，并调用之

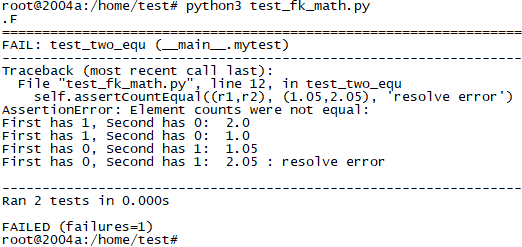
或者

python3 -m unittest test\_fk\_math.py

## 测试输出







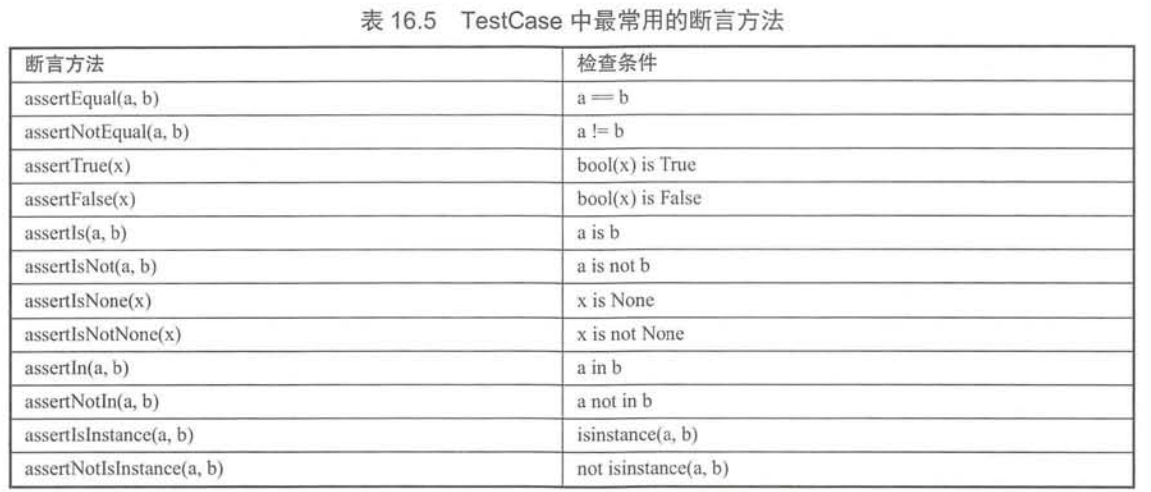
. 代表测试通过

F代表失败，failure

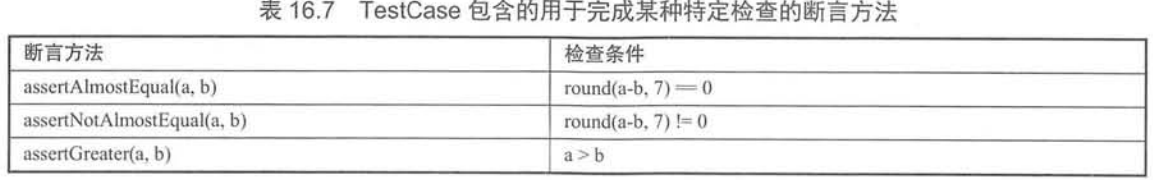
E代表出错，error

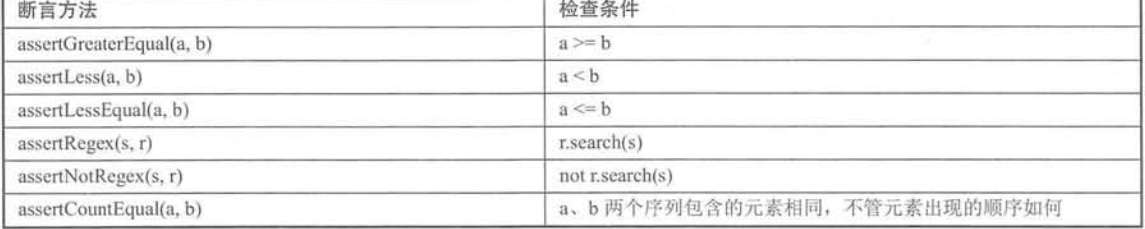
s代表跳过

## assertxxx方法





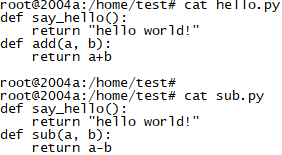


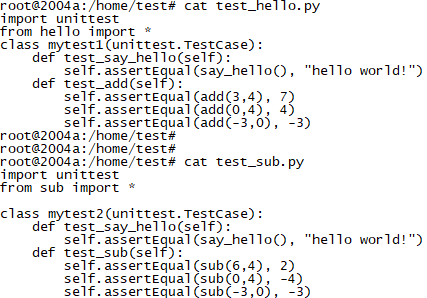


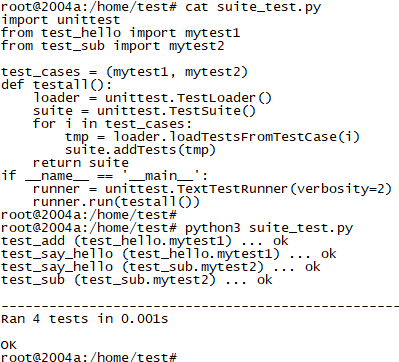


# 使用测试包

同时测试几个模块







verbosity可以生成更详细的测试信息

# 测试固件之setUp和tearDown

unittest除了有unittest.TestCase，TestSuite, TestRunner, 还有测试固件Test Fixture的概念

TestRunner是测试运行器，和TestSuite配合工作

unittest.TestCase包含了setUp()和tearDown()两个方法，

其中setUp用于初始化测试固件，tearDown用于销毁测试固件。

python3 -m unittest -v test\_hello.py #-v用于输出更详细的测试信息

## 对每个测试用例都setUp和tearDown

root@2004a:/home/learn# cat hello.py

def say\_hello():

return "hello world!"

def add(a, b):

return a+b

root@2004a:/home/learn# cat test\_hello.py

import unittest

from hello import \*

class mytest1(unittest.TestCase):

def test\_say\_hello(self):

self.assertEqual(say\_hello(), "hello world!")

def test\_add(self):

self.assertEqual(add(3,4), 7)

self.assertEqual(add(0,4), 4)

self.assertEqual(add(-3,0), -3)

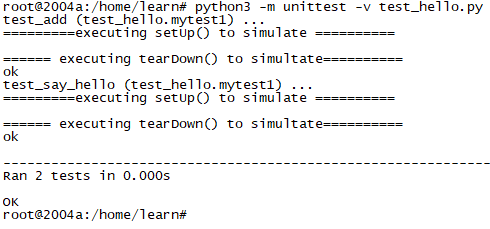
def setUp(self):

print("\n=========executing setUp() to simulate ==========")

def tearDown(self):

print("\n====== executing tearDown() to simultate==========")

root@2004a:/home/learn#



## 对某类只执行一次setUp和tearDown

对某类的所有测试用例只执行一次setUp和tearDown

root@2004a:/home/learn# cat test\_hello.py

import unittest

from hello import \*

class mytest1(unittest.TestCase):

def test\_say\_hello(self):

self.assertEqual(say\_hello(), "hello world!")

def test\_add(self):

self.assertEqual(add(3,4), 7)

self.assertEqual(add(0,4), 4)

self.assertEqual(add(-3,0), -3)

@classmethod

def setUpClass(cls):

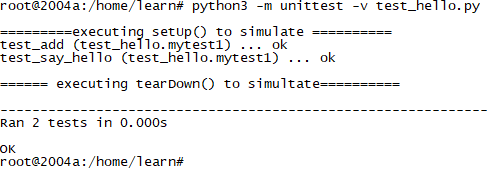
print("\n=========executing setUp() to simulate ==========")

@classmethod

def tearDownClass(cls):

print("\n====== executing tearDown() to simultate==========")

root@2004a:/home/learn#



# 跳过测试用例

有两种方法实现：使用装饰器，调用unittest.TestCase的skipTest 方法

@unittest.skip(reason)

@unittest.skipIf(condition, reason) #条件为真时执行

@unittest.skipUnless(conditiion, reason) #条件为假时执行

或

self.skipTest

## 使用skipxxx之前

root@2004a:/home/learn# cat hello.py

def say\_886():

return "886"

def say\_111():

return "111"

def say\_hello():

return "hello world!"

def add(a, b):

return a+b

root@2004a:/home/learn#

root@2004a:/home/learn# cat test\_hello.py

import unittest

from hello import \*

class mytest1(unittest.TestCase):

#@unittest.skip('临时跳过test\_say\_886')

def test\_say\_886(self):

self.assertEqual(say\_886(), "886")

def test\_say\_111(self):

#self.skipTest('临时跳过了test\_say\_111')

self.assertEqual(say\_111(), "111")

def test\_say\_hello(self):

self.assertEqual(say\_hello(), "hello world!")

def test\_add(self):

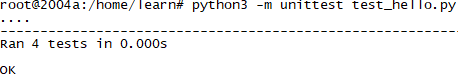
self.assertEqual(add(3,4), 7)

self.assertEqual(add(0,4), 4)

self.assertEqual(add(-3,0), -3)

root@2004a:/home/learn#

root@2004a:/home/learn#



## 使用skipxxx之后

