Содержание

1. Структура теста

Обычно библиотека для автоматизированного тестирования включает в себя следующие элементы:

Test Case

Модуль тестирования. Проверка валидности ответа для определенного набора входов

Test Suite

Набор ТестКейсов. Содержит в себе тесты, которые должны быть запущены вместе

Test Fixture

Содержит в себе предварительные настройки для тестов (например, соединение с БД, создание временных директорий)

Test Runner/Loader

Модуль, запускающий тесты и выводящий результаты в нужном формате (может предоставлять графический/текстовый интерфейс)

2. Основные методы TestCase

- *setUp()*
  Метод вызывается автоматически перед каждым методом в тесте. В нем производятся необходими предварительные настройки. По умолчанию не производит никаких настроек и изменений.
- *tearDown()*
  Метод вызывается автоматически после каждого метода в тесте, даже в тех случаях, когда тест не прошел. Метод будет вызван, если предварительный запуск setUp() прошел успешно.
- *run(result=None)*
  Запуск теста с автоматическим сбором результатов прохождения. При указании параметра *result* результаты будут записаны в него. Если же result не указан или None, то результаты будут записаны во временный объект.
- *skipTest(reason)*
  Пропуск теста. Параметр reason – причина пропуска.
- *debug()*
  Метод, используемый для отладки теста. При запуске, в отличие от run(), результат не собирается.
- *fail(msg=None)*
  Безусловный вызов сообщения о непрохождении теста. Парамертр msg – выдаваемое сообщение.
- *countTestCases()*
  Возвращает количество включенных в данный объект тестов. Для объектов класса TestCase это значение всегда будет равно 1.
- *id()*
  Возвращает строку, идентифицирующую тест.
- *doCleanups()*
  Метод автоматически вызывается при выполнении tearDown() или же в случае, когда setUp() выдал исключение. Метод удаляет созданные объекты и откатывает изменения.

Помимо этого, у TestCase есть множество методов для сравнения полученных в ходе теста значений с желаемыми.

В них обычно присутствуют параметры: a,b – первое и второе значения; msg – необязательный параметр, сообщение об ошибке.

Таблица 1 Методы точного сравнения

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

Неточное, «нечеткое» сравнение:

Таблица 2 Методы неточного сравнения

Method	Checks that	New in
assertAlmostEqual(a, b)	round(a-b, 7) == 0	
assertNotAlmostEqual(a, b)	round(a-b, 7) != 0	
assertGreater(a, b)	a > b	2.7

assertGreaterEqual(a, b)	a >= b	2.7
assertLess(a, b)	a < b	2.7
assertLessEqual(a, b)	a <= b	2.7
assertRegexpMatches(s, re)	regex.search(s)	2.7
assertNotRegexpMatches(s, re)	not regex.search(s)	2.7
assertItemsEqual(a, b)	sorted(a) == sorted(b) and works with unhashable objs	2.7
assertDictContainsSubset(a, b)	all the key/value pairs in *a* exist in *b*	2.7

Сравнение более сложных структур данных:

Таблица 3 Методы сравнения строк, списков, множеств

Method	Used to compare	New in
assertMultiLineEqual(a, b)	strings	2.7
assertSequenceEqual(a, b)	sequences	2.7
assertListEqual(a, b)	lists	2.7
assertTupleEqual(a, b)	tuples	2.7
assertSetEqual(a, b)	sets or frozensets	2.7
assertDictEqual(a, b)	dicts	2.7

Сравнение исключений:

Таблица 4 Сравнение исключений

Method	Checks that	New in
assertRaises(exc, fun, *args, **kwds)	fun(*args, **kwds) raises *exc*	
assertRaisesRegexp(exc, re, fun, *args, **kwds)	fun(*args, **kwds) raises *exc* and the message matches *re*	2.7

3. Основные методы TestSuite

- *addTest(test);* test – TestCase/TestSuite
  Добавление TestCase или TestSuite

- *addTests(tests)*
  Добавить набор TestCase/TestSuite
- *run(result)*
  Запустить набор тестов. Результат прохода тестов записывается в переменную result.
- *debug()*
  Метод для отладки тестов. Позволяет запустить тесты без сбора результата
- *countTestCases()*
  Возвращает количество включенных в тест подтестов
- *__iter__()*
  Итератор по тесту

4. Основные методы TestLoader

- *loadTestsFromTestCase(testCase)*
  Возвращает список тесткейсов, содержащихся в указанном testCase
- *loadTestsFromModule(module)*
  Возвращает список тесткейсов, содержащихся в указанном модуле
- *getTestCaseNames(testCase)*
  Возвращает последовательность имен тесткейсов
- discover(*start_dir*, *pattern='test*.py'*, *top_level_dir=None*)
  Ищет и возвращает список тестовых модулей в указанной директории и ее поддиректориях, соответствующих переданному шаблону (если есть).
- *sortTestMethodsUsing()*
  Функция для сравнения названий тестов при их сортировке в getTestCaseNames(testCase)
- *suiteClass*
  Вызываемый объект, конструирующий test suite из списка тестов.

5. Основные методы TestResult

- *Errors*
  Список ошибок. Кортеж, состоящий из тесткейса и сработавшего исключения
- *Failures*
  Список неудачно отработавших тестов.
- *Skipped*
  Список пропущенных тестов, включающий в себя название теста и причину
- *expectedFailures*
  Список ожидаемых «падений» тестов.
- *unexpectedSuccesses*
  Список тестов, которые должны были выдать ошибку, но неожиданно отработали
- *shouldStop*
  Должен быть установлен в True, если запуск тестов должен быть прерван методом stop()

- *testsRun*
  Количество запущенных тестов
- *failfast*
  Если установлено в True, то метод stop() будет вызван после первого неудачно отработавшего теста
- *stop()*
  Остановка запущенных тестов
- *startTest(test)*
  Вызывается, когда тест test должен быть запущен
- *stopTest(test)*
  Вызывается, когда тест test должен быть остановлен
- *startTestRun(test)*
  Вызывается перед запуском всех тестов
- *stopTestRun(test)*
  Вызывается перед остановкой всех тестов
- *addFailure(test,err)*
  Вызывается при «падении» теста. Добавление «падение» в кортеж (тест, падение)
- *addSuccess(test)*
  Вызывается при удачном прохождении теста. По умолчанию ничего не делает.
- *addSkip(test,reason)*
  Вызывается, когда тест пропущен. Добавляет запись (тест, причина пропуска)
- *addExpectedFailure(test,err)*
  Вызывается при ожидаемом «падении» теста. Добавляет запись (тест, ошибка)
- *addUnexpectedSuccess(test)*
  Вызывается при неожиданном прохождении теста.

6. Создание теста

Предположим, у нас есть класс Животное (умеющее говорить):

```
class Animal:
    def speak(self):
        pass
```

От него наследуются два класса – Кошка и Собака (у них имеются клички и говорят они по-разному):

```
class Cat(Animal):
    def __init__(self, name):
        self.name = name
    def speak(self):
        return "Meow"
class Dog(Animal):
    def __init__(self, name):
```

```
        self.name = name
    def speak(self):
        return "Wow"
```

Давайте протестируем:

a. Создание объекта
b. Их способность говорить (метод speak)

В директории, где расположен файл с описанием классов, создадим еще один файл (назовем его, допустим, AnimalTest)

Сначала импортируем модель тестирования:

```
from django.utils import unittest
```

Помимо этого, нам понадобятся и наши созданные классы животных:

```
from animal import *
```

Создадим класс AnimalTestCase, являющийся наследником unittest.TestCase:

```
class AnimalTestCase(unittest.TestCase):
```

Определим метод предварительных операций (в нем мы будем каждый раз создавать объекты классов Cat и Dog):

```
    def setUp(self):
        self.cat = Cat('Tiger')
        self.dog = Dog('Bingo')
```

Определим метод для проверки метода speak:

```
    def testSpeaking(self):
        self.assertEqual(self.cat.speak(), "Meow") # проверка равенства
        self.assertEqual(self.dog.speak(), "Wow")
```

Определим метод для проверки правильности назначения имени:

```
    def testName(self):
        self.assertEqual(self.cat.name, 'Tiger')
        self.assertEqual(self.dog.name, 'Bingo')
```

Запуск тестов:

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

Попробуем запустить:

```
C:\Users\ivri\BitNami DjangoStack projects\sort>python animalTest.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.003s

OK
```

Рис. 1 Результат тестов

7. Изменение детализации вывода

Изменить подробность выводимой информации можно следующим образом:
Заменим строку

unittest.main()

на

suite = unittest.TestLoader().loadTestsFromTestCase(AnimalTestCase)
unittest.TextTestRunner(verbosity=2).run(suite)

Здесь мы сначала загружаем тесты из указанного тесткейса в TestLoader, а затем передаем его в TestRunner. Степень подробности – verbosity – указываем равной 2.

В результате количество выводимой информации становится существенно больше:

```
C:\Users\ivri\BitNami DjangoStack projects\sort>python animalTest.py
testName (__main__.AnimalTestCase) ... ok
testSpeaking (__main__.AnimalTestCase) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.009s

OK
```

Рис. 2 Результат тестов с более детальной информацией

8. Вывод ошибок в тестах

Посмотрим, что получится, если мы намеренно сделаем ошибку:

```
class Cat(Animal):
    def speak(self):
        return "Gav"
```

```
C:\Users\ivri\BitNami DjangoStack projects\sort>python animalTest.py
testName (__main__.AnimalTestCase) ... ok
testSpeaking (__main__.AnimalTestCase) ... FAIL

======================================================================
FAIL: testSpeaking (__main__.AnimalTestCase)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "animalTest.py", line 10, in testSpeaking
    self.assertEqual(self.cat.speak(), "Meow")
AssertionError: 'Gav' != 'Meow'

----------------------------------------------------------------------
Ran 2 tests in 0.036s

FAILED (failures=1)
```

Рис. 3 Вывод ошибок

Будет выведено имя несработавшего метода, а также конкретная строка ошибки.

9. Немного полезностей при использовании TestSuite

Давайте рассмотрим еще немного полезных возможностей TestSuite.
Во-первых, он поддерживает итерацию. Добавим к предыдущим моджификациям строки:

```
for test in suite:
        print test
```

Данный код вывод список тестов :

```
C:\Users\ivri\BitNami DjangoStack projects\sort>python animalTest.py
testName (__main__.AnimalTestCase)
testSpeaking (__main__.AnimalTestCase)
```

Рис. 4 Вывод списка тестов

Тесты можно добавлять и по одному:

```
suite = unittest.TestSuite()
suite.addTest(AnimalTestCase('testSpeaking'))
suite.addTest(AnimalTestCase('testName'))
```

10. Пропуск тестов

Некоторые тесты можно умышленно пропускать. Для этого перед определением теста нужно добавить атрибут @unittest.skip:

```
@unittest.skip("demonstrating skipping")
  def testName(self): …
```

```
C:\Users\ivri\BitNami DjangoStack projects\sort>python animalTest.py
testName (__main__.AnimalTestCase) ... skipped 'demonstrating skipping'
testSpeaking (__main__.AnimalTestCase) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.009s

OK (skipped=1)
```

Рис. 5 Пропуск тестов

Помимо этого, используются:

- Пропуск с условием:
  @unittest.skipIf(mylib.__version__ < (1, 3), "not supported in this library version")
  @unittest.skipUnless(sys.platform.startswith("win"), "requires Windows")
- Пропуск всего класса :
  @skip("showing class skipping")
  class MySkippedTestCase(unittest.TestCase)

11. Тестирование модели в Django

Воспользуемся моделью из предыдущей лабораторной работы. При создании модели автоматически был создан файл tests.py следующего содержания:

```
from django.test import TestCase
class SimpleTest(TestCase):
  def test_basic_addition(self):
    """
     Tests that 1 + 1 always equals 2.
     """
```

```
self.assertEqual(1 + 1, 2)
```

В нем приведен пример самого простого теста на сложение двух целых чисел. Заметим, что в данном случае родительский класс *TestCase* импортируется из *django.test*.

Давайте в этом примере напишем тест для класса Publisher:

```
class Publisher(models.Model):
    name = models.CharField(max_length=30)
    address = models.CharField(max_length=50)
    city = models.CharField(max_length=60)
    state_province = models.CharField(max_length=30)
    country = models.CharField(max_length=50)
    website = models.URLField()
    def __unicode__(self):
        return self.name
```

Протестируем:

- Создание объекта
- Строковое представление объекта
- Сохранение объекта
- Удаление объекта

Сначала создадим класс-наследник TestCase:

```
class PublisherTestCase(TestCase):
```

В него добавим два метода, которые будут автоматически вызываться до(setUp – создает объект) и после(tearDown – удаляет объект) каждого метода-теста.

```
    def setUp(self):
        self.pub1=Publisher.objects.create(name="Drofa",
address="Krasnopresnensyaya, 1",
city="Moscow",state_province="Moscow",country="Russia",
website="drofa.ru")
    def tearDown(self):
        self.pub1=None
```

А теперь добавим методы проверки строкового представления (*testObjectAsString*), сохранения (*testSaveObject*) и удаления (*testDeleteObject*) объекта. Заметим, что все методы начинаются с *test* : метод, названный таким образом автоматически распознается как тест (если предварительно не изменить настройки).

```
    def testObjectAsString(self):
        self.assertEqual(str(self.pub1), 'Drofa')

    def testSaveObject(self):
        self.pub1.save()
        obj=Publisher.objects.get(name="Drofa")
```

```
            self.assertEqual(obj.name, 'Drofa')

        def testDeleteObject(self):
            self.pub1.save()
            Publisher.objects.filter(name="Drofa").delete()
self.assertNotEqual(Publisher.objects.filter(name="Drofa"),'Drofa')
```

Для запуска тестирования модели нужно перейти в каталог проекта (не модели!) и запустить скрипт

python manage.py test

12. Тестирование шаблонов

В Django, помимо всего прочего, поддерживается возможность тестирования шаблонов.

Допустим, мы хотим протестировать шаблон *books.html*.

Для этого нам понадобится дополнительный элемент Client, который будет имитировать работу браузера:

from django.test.client import Client

Создадим класс для тестирования шаблона:

class TemplateTestCase(TestCase):

В предварительных настройках создадим клиента:

def setUp(self):

self.client=Client()

Метод для тестирования шаблона

def testBooksPage(self):

Запрашиваем данные по адресу /books/:

response=self.client.get("/books/")

Проверяем наличие поля Author и статус код:

self.assertContains(response, 'Author',status_code=200)

Проверяем, что нужный шаблон был вызван:

self.assertTemplateUsed(response, 'books.html')

После окончания теста удаляем клиента:

def tearDown(self):

self.client=None

13. Задание

Для модели, описанной в предыдущей лабораторной работе написать следующие тесты:

1. Тест для определенного класса модели
2. Тест для шаблона
3. В отчет: основные скриншоты, описания тестов и код программы.

Более полная документация - http://docs.python.org/library/unittest.html (на английском)