**Разновидности тестирования**

*Черный ящик* - Не известно ничего

*Серый ящик* - Известны детали реализации или доступны описания интерфейсов

*Белый ящик* - Доступна любая необходимая информация, включая исходный код

**Какие бывают тесты**

*Юнит тесты* - проверяют работу отдельных компонент системы

(функций и методов классов в рамках одного модуля)

*Интеграционные тесты* - проверка работоспособности комплексов модулей и/или всей системы

**doctest**

doctest\_example.py:

**def** kth\_stat(iterable, k):  
 *"""Compute k-order stat in iterable  
 >>> kth\_stat([4, 1, 0], 1)  
 0  
 >>> kth\_stat(range(100), 10)  
 10  
 """* **return** sorted(iterable)[k - 1]

python -m doctest doctest\_example.py

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

File "/Users/artem/Desktop/python\_programs/sqlalchemy/doctest\_example.py", line 5, in doctest\_example.kth\_stat

Failed example:

kth\_stat(range(100), 10)

Expected:

10

Got:

9

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

1 items had failures:

1 of 2 in doctest\_example.kth\_stat

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

doctest: полезные директивы

SKIP - не запускает тесты на помеченном примере

IGNORE\_EXCEPTION\_DETAIL - игнорирует текст эксепшна

ELLIPSIS - позволяет написать … вместо любого несущественного для проверки вывода теста

FAIL\_FAST - останавливается после первого упавшего теста

Подробности по ссылке:

https://docs.python.org/3/library/doctest.html#doctest-directives

<https://docs.python.org/3/library/doctest.html>

Пример использования директив

**def** kth\_stat(iterable, k):  
 *"""Compute k-order stat in iterable  
 >>> kth\_stat(range(10000), 10000) # doctest: +ELLIPSIS  
 9...9  
 >>> kth\_stat(1, 1) # doctest: +IGNORE\_EXXEPTION\_DERAIL  
 Traceback (most recent call last):  
 ...  
 AssertionError: blablabla  
 """* **assert** isinstance(iterable, Iterable), **"expected iterable as first argument"  
 return** sorted(iterable)[k - 1]

**unittest**

**import** unittest  
**import** random  
**from** doctest\_example **import** kth\_stat  
  
  
**class** TestStat(unittest.TestCase):  
 **def** test\_on\_range(self):  
 **assert** kth\_stat(range(10), 3) == 2 *# don't do it* **def** test\_on\_shuffled\_range(self):  
 li = list(range(10))  
 random.shuffle(li)  
 self.assertEqual(kth\_stat(li, 3), 2)

Запуск unittest

**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 unittest.main()

python -m unittest test\_doctest\_example.py

| **Method** | **Checks that** | **New in** |
| --- | --- | --- |
| [assertEqual(a, b)](https://docs.python.org/3/library/unittest.html" \l "unittest.TestCase.assertEqual) | a == b |  |
| [assertNotEqual(a, b)](https://docs.python.org/3/library/unittest.html" \l "unittest.TestCase.assertNotEqual) | a != b |  |
| [assertTrue(x)](https://docs.python.org/3/library/unittest.html" \l "unittest.TestCase.assertTrue) | bool(x) is True |  |
| [assertFalse(x)](https://docs.python.org/3/library/unittest.html" \l "unittest.TestCase.assertFalse) | bool(x) is False |  |
| [assertIs(a, b)](https://docs.python.org/3/library/unittest.html" \l "unittest.TestCase.assertIs) | a is b | 3.1 |
| [assertIsNot(a, b)](https://docs.python.org/3/library/unittest.html" \l "unittest.TestCase.assertIsNot) | a is not b | 3.1 |
| [assertIsNone(x)](https://docs.python.org/3/library/unittest.html" \l "unittest.TestCase.assertIsNone) | x is None | 3.1 |
| [assertIsNotNone(x)](https://docs.python.org/3/library/unittest.html" \l "unittest.TestCase.assertIsNotNone) | x is not None | 3.1 |
| [assertIn(a, b)](https://docs.python.org/3/library/unittest.html" \l "unittest.TestCase.assertIn) | a in b | 3.1 |
| [assertNotIn(a, b)](https://docs.python.org/3/library/unittest.html" \l "unittest.TestCase.assertNotIn) | a not in b | 3.1 |
| [assertIsInstance(a, b)](https://docs.python.org/3/library/unittest.html" \l "unittest.TestCase.assertIsInstance) | isinstance(a, b) | 3.2 |
| [assertNotIsInstance(a, b)](https://docs.python.org/3/library/unittest.html" \l "unittest.TestCase.assertNotIsInstance) | not isinstance(a, b) | 3.2 |

**Фикстуры**

Тесты могут требовать особой настройки окружения, подготовки данных или выполнения каких-то других действий перед или после запуска теста или модуля с тестами.

*Фикстуры* - функции, вызываемые до или после теста для выполнения таких настроек.

Методы setUp() и tearDown() (которые в данном простом случае не нужны) позволяют определять инструкции, выполняемые перед и после каждого теста, соответственно.

**class** TestWithTempFile(unittest.TestCase):  
 **def** setUp(self):  
 self.tempfile = tempfile.TemporaryFile(mode=**'w+'**)  
 li = list(range(10000))  
 random.shuffle(li)  
 json.dump(li, self.tempfile)  
 self.tempfile.seel(0)  
  
 **def** tearDown(self):  
 self.tempfile.close()  
  
 **def** test\_on\_large\_seq\_from\_file(self):  
 self.assertEqual(kth\_stat(  
 json.load(self.tempfile, 300), 299))

unittest.mock

Иногда необходимо изолировать часть программы для того, чтобы тестировать только минимально возможную часть системы, для этого можно использовать специальные объекты, подменяющие внешние объекты или функции.

unittest.mock - это набор универсальных объектов для таких подмен.

**pytest**

pip install -U pytest

first\_test.py:

**import** pytest  
  
**def** func(x):  
 **return** x + 5  
  
**def** test\_method():  
 **assert** func(3) == 8

pytest first\_test.py

multiple\_test.py:

**import** pytest  
  
**def** test\_method1():  
 x = 5  
 y = 10  
 **assert** x == y  
  
**def** test\_method2():  
 a = 15  
 b = 20  
 **assert** a+5 == b

py.test -k method2 -v

**import** pytest  
  
  
@pytest.mark.one  
**def** test\_method1():  
 x = 5  
 y = 10  
 **assert** x == y  
  
  
@pytest.mark.two  
**def** test\_method2():  
 a = 15  
 b = 20  
 **assert** a+5 == b

py.test -m one

kth\_stat.py:

**def** kth\_stat(iterable, k):  
 *"""Compute k-order stat in iterable  
 >>> kth\_stat([4, 1, 0], 1)  
 0  
 >>> kth\_stat(range(100), 10)  
 10  
 """* **return** sorted(iterable)[k - 1]

test\_kth\_stat.py:

**import** random  
**from** kth\_stat **import** kth\_stat  
  
  
**def** test\_on\_range():  
 **assert** kth\_stat(range(10), 3) == 2  
  
  
**def** test\_on\_shuffled\_range():  
 li = list(range(10))  
 random.shuffle(li)  
 **assert** kth\_stat(li, 3) == 2

Fixtures

**import** pytest  
  
@pytest.fixture  
**def** filled\_file():  
 **with** tempfile.TemporaryFile(mode=**'w+'**) **as** f:  
 li = list(range(10000))  
 random.shuffle(li)  
 json.dump(li, self.tempfile)  
 self.tempfile.seel(0)  
 **yield** f  
  
**def** test\_on\_large\_seq\_from\_file(filled\_file):  
 **assert** kth\_stat(json.load(filled\_file), 300) == 299

**import** pytest  
  
@pytest.fixture  
**def** some\_common\_dependency():  
 print(**"setup some very useful stuff"**)  
  
@pytest.fixture  
**def** rare\_dependency(some\_common\_dependency):  
 print(**"rare\_dependency"**)  
  
@pytest.fixture  
**def** rare\_dependency\_for\_test\_one(rare\_dependency):  
 print(**"rare\_dependency\_for\_test\_one"**)  
  
**def** test\_one(rare\_dependency\_for\_test\_one):  
 print(**"test one"**)  
  
**def** test\_two(rare\_dependency):  
 print(**"test two"**)

*conftest.py* - автозагружаемый файл с фиксутрами.

Фикстуры, опции и хуки из этого файла автоматически становятся доступными во всех тестах.

Полезные опции

--collect-only - вывод списка найденных тестов

-k - фильтрации по имени теста

-s - включает вывод stdout & stderr тестов (по умолчанию выводятся только для упавших тестов)

-v - повышает детализацию процесса запуска тестов

--lf, --last-failed - перезапускает тесты, упавшие при последнем запуске

--sw, --stepwise - выходит при падении и при последующих запусках, продолжает с последнего упавшего теста

pytest.ini - основной конфигурационный файл, в котором можно изменять поведение pytest по умолчанию

[pytest]

adopts = --cov-report=html --cov<path> --flake8

testpaths = <test\_paths>