Лекция 11: Тестирование

Сергей Лебедев

sergei.a.lebedev@gmail.com

23 ноября 2015 г.

Тестировать или **не** тестировать?[1]

- Тестировать:
  - тесты проверяют корректность кода
  - и позволяют бесстрашно изменять код даже в больших проектах.
- Не тестировать:
  - написание тестов требует времени,
  - нередко в проекте тестов больше чем кода,
  - работающие тесты **не гарантируют** корректность.
- Тем не менее, ответ очевиден: конечно же тестировать!

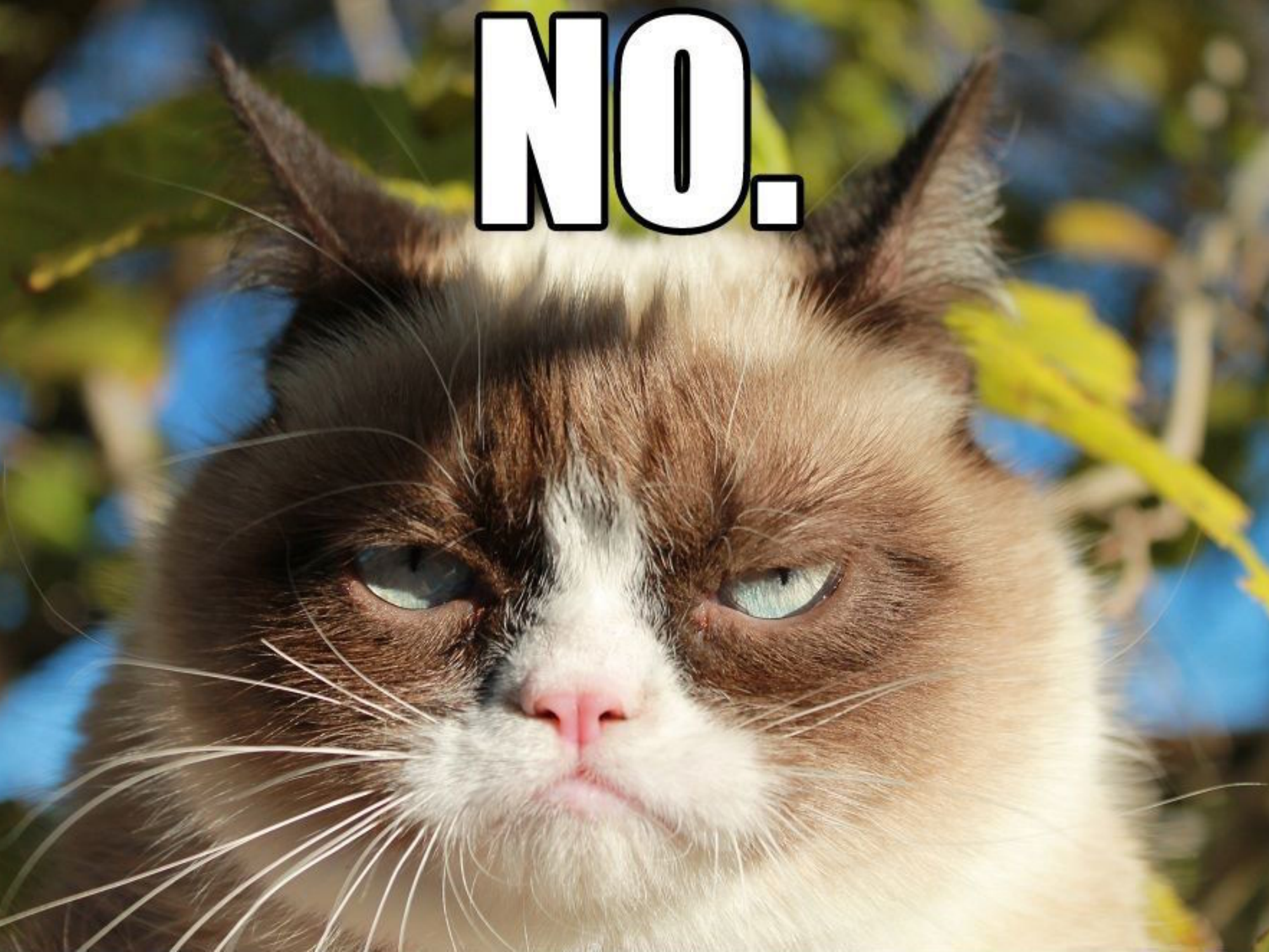
[1] Здесь и далее под тестированием имеется в виду модульное (и иногда интеграционное) тестирование *aka* unit testing.

```
>>> import itertools
>>> def rle(iterable):
...     """Applies run-length encoding to an iterable."""
...     for item, g in itertools.groupby(iterable):
...         yield item, sum(1 for _ in g)
...
>>> list(rle("mississippi"))
[('m', 1), ('i', 1), ('s', 2), ('i', 1),
 ('s', 2), ('i', 1), ('p', 2), ('i', 1)]
```

Тестирование с помощью `print`

- Функция `print` позволяет организовать полуавтоматическое тестирование вне интерактивной оболочки.
- Интерпретатор печатает — вы проверяете:

```
>>> def test_rle():
...     print(list(rle("mississippi")))
...
>>> test_rle()
[('m', 1), ('i', 1), ('s', 2), ('i', 1),
 ('s', 2), ('i', 1), ('p', 2), ('i', 1)]
```


NO.

Модуль doctest

```python
import doctest
import itertools

def rle(iterable):
    """Applies run-length encoding to an iterable.

    >>> list(rle(""))
    []
    >>> list(rle("mississippi"))
    [('m', 1), ('i', 1), ('s', 2), ('i', 1),
     ('s', 2), ('i', 1), ('p', 2), ('i', 1)]
    """
    for item, g in itertools.groupby(iterable):
        yield item, sum(1 for _ in g)


if __name__ == "__main__":
    doctest.testmod()
```

[2] https://docs.python.org/3/library/doctest

```
$ python ./test_doctest.py  # можно python -m doctest
**********************************************************************
File "test_doctest.py", line 10, in __main__.rle
Failed example:
    list(rle("mississippi"))
Expected:
    [('m', 1), ('i', 1), ('s', 2), ('i', 1),
     ('s', 2), ('i', 1), ('p', 2), ('i', 1)]
Got:
    [('m', 1), ('i', 1), ('s', 2), ('i', 1), ...]
**********************************************************************
1 items had failures:
   1 of   2 in __main__.rle
***Test Failed*** 1 failures.
```

- Директивы позволяют изменить то, как `doctest` сравнивает ожидаемый вывод интерпретатора с фактическим.
- Например, директива `NORMALIZE_WHITESPACE` нормализует пробельные символы перед сравнением:

```
>>> list(rle("mississippi"))
... # doctest: +NORMALIZE_WHITESPACE
[('m', 1), ('i', 1), ('s', 2), ('i', 1),
 ('s', 2), ('i', 1), ('p', 2), ('i', 1)]
```

- А директива `ELLIPSIS` позволяет использовать символ `...`, который совпадает с любой строкой:

```
>>> list(rle("mississippi"))
... # doctest: +ELLIPSIS
[('m', 1), ('i', 1), ('s', 2), ('i', 1), ...]
```

- Модуль `doctest` позволяет проверить реализацию функции на соответствие записанному сеансу интерпретатора.
- Плюсы:
  - доступен в стандартной библиотеке,
  - решает задачу тестирования для небольших проектов,
  - доктесты их легко читать,
  - примеры кода в документации всегда актуальны.
- Минусы:
  - доктесты требуют, чтобы у результата было содержательное строковое представление,
  - длинные доктесты ухудшают читаемость документации,
  - нет способа запустить подмножество доктестов,
  - если в середине доктеста произошла ошибка, оставшаяся часть не выполнится.

assert

- Напоминание:
  - оператор **assert** принимает два аргумента: условие и произвольное значение,
  - если условие falsy, оператор поднимает исключение **AssertionError**.

```
>>> assert [], 42
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError: 42
```

- Протестируем функцию `rle`:

```
>>> def test_rle():
...     s = "mississippi"
...     tmp = set(ch for ch, _count in rle(s))
...     assert tmp == set(s[:-1] + s[1])
...     assert not list(rle(""))
...
>>> test_rle()
```

- Вопрос: что вы думаете про такой тест?

- Хороший тест:
  - корректный,
  - понятный читателю,
  - конкретный, то есть проверяет что-то одно.
- Попробуем улучшить тест для функции `rle`:

```
>>> def test_rle():
...     assert rle("mississippi") == [
...         ('m', 1), ('i', 1), ('s', 2), ('i', 1),
...         ('s', 2), ('i', 1), ('p', 2), ('i', 1)
...     ]
...
>>> def test_rle_empty():
...     assert not list(rle(""))
...
>>> test_rle_empty()
>>> test_rle()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 4, in test_rle
AssertionError
```

Второй аргумент оператора **assert** используется для сообщения об ошибке:

```
>>> def test_rle():
...     actual = rle("mississippi")
...     expected = [
...         ('m', 1), ('i', 1), ('s', 2), ('i', 1),
...         ('s', 2), ('i', 1), ('p', 2), ('i', 1)
...     ]
...     message = "{} != {}".format(actual, expected)
...     assert actual == expected, message
...
>>> test_rle()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 8, in test_rle
AssertionError: <generator object rle at [...]> != \
    [('m', 1), ('i', 1), ('s', 2), ...]
```

- Добавим сообщение об ошибке к тесту `test_rle_empty`:

```
def test_rle_empty():  # ^C/^V
    actual = list(rle(""))
    expected = []
    message = "{} != {}".format(actual, expected)
    assert actual == expected, message
```

- Самое время вынести логику сравнения в отдельную функцию:

```
def assert_equal(x, y):
    assert x == y, "{} != {}".format(x, y)
```

- Вопрос: что делать, если мы также хотим проверять утверждения вида `("a", 2) in rle(...)`?

- Оператор **`assert`** можно использовать для написания тестов.
- Плюсы:
  - тесты с **`assert`** легко читать,
  - они не используют ничего кроме стандартных средств языка,
  - в отличие от доктестов это обычные функции.
- Минусы:
  - запускать тесты нужно вручную,
  - их сложно отлаживать, потому что
  - для каждого типа утверждения приходится самостоятельно конструировать сообщение об ошибке.

Модуль unittest

- Модуль `unittest` реализует функциональность JUnit для тестирования кода на Python.
- Наследие Java до сих пор в обилии присутствует в API.
- Перепишем имеющиеся тесты с использованием `unittest`:

```
import unittest

class TestHomework(unittest.TestCase):
    def test_rle(self):
        self.assertEqual(rle("mississippi"), [...])

    def test_rle_empty(self):
        self.assertEqual(list(rle("")), [])


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

[3] https://docs.python.org/3/library/unittest

```
$ python ./test_homework.py
F.
======================================================================
FAIL: test_rle (__main__.TestHomework)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "./test_homework.py", line 12, in test_rle
    self.assertEqual(rle("mississippi"), expected)
AssertionError: <generator object rle at [...]> != \
    [('m', 1), ('i', 1), ('s', 2), ...]

----------------------------------------------------------------------
Ran 2 tests in 0.001s

FAILED (failures=1)
```

- Функция `unittest.main` загружает все тесты текущего модуля и запускает их.
- Тест — метод экземпляра `unittest.TestCase`, начинающийся на `test`.
- При необходимости тесты можно объединять в группы с помощью класса `unittest.TestSuite`:

```
suite = unittest.TestSuite([
    TestHomework(),
    TestSomethingElse()
])
```

- Указывать вручную, что нужно запустить довольно досадно.
- Вопрос: можно ли лучше?

```
#                          V в текущей директории
$ python -m unittest .
F.
======================================================================
FAIL: test_rle (test_homework.TestHomework)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "./test_homework.py", line 12, in test_rle
    self.assertEqual(rle("mississippi"), expected)
AssertionError: <generator object rle at [...]> != \
    [('m', 1), ('i', 1), ('s', 2), ...]

----------------------------------------------------------------------
Ran 2 tests in 0.001s

FAILED (failures=1)
```

Методы unittest.TestCase

```
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
assertIsNot(a, b)              a is not b
assertIsNone(x)                x is None
assertIsNotNone(x)             x is not None
assertIn(a, b)                 a in b
assertNotIn(a, b)              a not in b
assertIsInstance(a, b)         isinstance(a, b)
assertNotIsInstance(a, b)      not isinstance(a, b)

assertRaises(exc_type)         # ?
```

- ? *aka* fixtures — способ подготовить контекст, в котором будут запускаться тесты.
- Это можно использовать, например, для работы с ресурсами: сокетами, файлами, временными директориями.
- Пример:

```
class TestHomeworkWithOracle(unittest.TestCase):
    def setUp(self):
        self.oracle = RleOracle("http://oracle.rle.com")

    def test_rle_against_oracle(self):
        s = "mississippi"
        self.assertEqual(list(rle(s)), self.oracle(s))

    def tearDown(self):
        self.oracule.close()
```

Модуль `unittest`: резюме

- Модуль `unittest` — клон JUnit для Python.
- Мы обсудили
  - основные сущности `unittest`,
  - как писать и запускать тесты,
  - какие полезные методы имеются в классе `unittest.TestCase`.
- Плюсы:
  - доступен в стандартной библиотеке,
  - выводит понятные сообщения об ошибках,
  - умеет автоматически находить тесты.
- Минусы:
  - API унаследован от Java,
  - заставляет писать много лишнего кода,
  - читать `unittest` тесты сложнее, чем доктесты и тесты, использующие **`assert`**.

Пакет `py.test`

- Пакет `py.test` — популярная альтернатива `unittest` для написания и запуска тестов.
- Отличительная особенность `py.test` — практически полное отсутствие API: тесты можно писать, используя стандартные средства языка.

```
def test_rle():
    assert rle("mississippi") == [
        ('m', 1), ('i', 1), ('s', 2), ('i', 1),
        ('s', 2), ('i', 1), ('p', 2), ('i', 1)
    ]


def test_rle_empty():
    assert not list(rle(""))
```

[4] https://pytest.org


YES!

```
$ python -m pytest -q test_pytest.py
F.
======================== FAILURES ========================
_____________________ test_rle _____________________

    def test_rle():
>       assert rle("mississippi") == [
            ('m', 1), ('i', 1), ('s', 2), ('i', 1),
            ('s', 2), ('i', 1), ('p', 2), ('i', 1)
        ]
E       assert <generator ...> == [('m', 1), ..., ('i', 1)]
E         Full diff:
E         - <generator object rle at [...]>
E         + [('m', 1),
E         +  ('i', 1),
E             ...
E         +  ('i', 1)]

test_pytest.py:5: AssertionError
```

Пакет py.test: интерфейс командной строки

- Запустить py.test можно 1001 способом, например:
  - найти тесты в текущей директории и во всех вложенных директориях и запустить их:
    ```
    $ python -m pytest
    ```
  - найти и запустить тесты в указанном файле:
    ```
    $ python -m pytest test_pytest.py
    ```
  - запустить один тест в файле по имени:
    ```
    $ python -m pytest test_pytest.py::test_rle
    ```
- Что такое тест для py.test[5]?
  - функция test_*,
  - метод test_* в классе Test* или в классе, наследующемся от unittest.TestCase,
  - доктест, если py.test был запущен с параметром --doctest-modules.

[5] https://pytest.org/latest/goodpractises.html

```
FF
========================== FAILURES ==========================
_________________________ test_in _________________________

    def test_in():
>       assert 42 in range(0, 10)
E       assert 42 in range(0, 10)

test_pytest_introspection.py:3: AssertionError
______________________ test_in_range_ ______________________

    def test_in_range():
        x = 42
>       assert x > 10 and x < 15
E       assert (42 > 10 and 42 < 15)

test_pytest_introspection.py:8: AssertionError
```

```
def test_undo_dict_exceptions():
    d = UndoDict()
    with pytest.raises(KeyError):
        d["foo"]
```

```
F
========================= FAILURES ========================
___________________ test_undo_dict_exceptions ____________

    def test_undo_dict_exceptions():
        d = UndoDict()
        with pytest.raises(KeyError):
>           d["foo"]
E           Failed: DID NOT RAISE

test_assert_raises.py:15: Failed
```

Возможности py.test: параметрические тесты

```
def cut_suffix(s, suffix):
    return s[:s.rfind(suffix)]


.F.
========================== FAILURES ==========================
__________ test_cut_suffix[foobar-boo-foobar] __________

    @pytest.mark.parametrize("s,suffix,expected", [
        ("foobar", "bar", "foo"),
        ("foobar", "boo", "foobar"),
        ("foobarbar", "bar", "foobar")
    ])
    def test_cut_suffix(s, suffix, expected):
>       assert cut_suffix(s, suffix) == expected
E       assert 'fooba' == 'foobar'
E         - fooba
E         + foobar
E         ?      +

test_parametric_tests.py:14: AssertionError
```

- py.test реализует аналоги TestCase.setUp и TestCase.tearDown,
- но это не самая интересная его возможность.
- Логику подготовки контекста конкретного типа можно абстрагировать

```
@pytest.yield_fixture
def oracle(request):
    oracle = RleOracle("http://oracle.rle.com")
    yield oracle
    oracle.close()
```

- чтобы потом неявно использовать в тестах:

```
def test_rle_against_oracle(oracle):
    s = "mississippi"
    assert list(rle(s)) == oracle(s)
```

```
import io
import gzip
import sys


def test_ook_eval(capsys, monkeypatch):
    handle = io.StringIO("!")
    monkeypatch.setattr(sys, "stdin", handle)
    ook_eval("Ook. Ook! Ook! Ook.")
    output, _err = capsys.readouterr()
    assert output == "!"


def test_reader(tmpdir):
    path = tmpdir.join("example.gz")
    path.write("")
    assert isinstance(reader(str(path)), gzip.GzipFile)
```

- Пакет `py.test` — швейцарский нож тестирования в мире Python.
- Плюсы:
  - практически нет API, тесты — обычные функции,
  - удобный вывод,
  - удобный механизм параметризации тестов,
  - приспособления, которые можно переиспользовать,
  - множество встроенных возможностей и впечатляющее количество расширений[6].
- Минусы:
  - магия-магия-магия,
  - может быть сложнее для понимания, если вы привыкли к JUnit.

[6] https://pytest.org/latest/plugins_index

Пакет `hypothesis`

- До текущего момента мы обсуждали тесты, которые проверяют тривиальные свойства кода:
  - равенство ожидаемому значению,
  - вхождение результата в коллекцию и так далее.
- Иногда можно формулировать и проверять менее тривиальные свойства, например:
  - если `sorted` возвращает список, отсортированный по неубыванию,
  - то для любого списка `xs` и индексов `i < j` справедливо, что `sorted(xs)[i] <= sorted(xs)[j]`.
- Вопрос: как проверить это свойство?

```
import random

def random_array():
    size = random.randint(0, 1024)
    return [random.randint(-42, 42) for _ in range(size)]


def test_sort():
    xs = random_array()
    result = sorted(xs)
    assert all(xi <= xj
               for xi, xj in zip(result, result[1:]))
```

Такой подход сработает, но

- писать генераторы самостоятельно долго и бессмысленно,
- на практике нас, как правило, интересует минимальный контрпример.

- Пакет `hypothesis` реализует API для формулирования и проверки свойств.
- Перепишем `test_sort` с использованием `hypothesis`:

```
import hypothesis.strategies as st
from hypothesis import given

@given(st.lists(st.integers()))
def test_sort(xs):
    result = sorted(xs)
    assert all(xi <= xj
               for xi, xj in zip(result, result[1:]))
```

- И попробуем обмануть его:

```
def sorted(xs, f=sorted):
    return xs if len(xs) == 8 else f(xs)
```

[7] https://hypothesis.readthedocs.org

```
F
========================== FAILURES ===========================
_________________________ test_sort __________________________
    assert all(xi <= xj
E   assert all(<generator object <genexpr> at 0x103b81e10>)
----------------------- Hypothesis -----------------------
Falsifying example: test_sort(xs=[0, 0, 0, 1, 0, 0, 0])
```

Пакет hypothesis: что можно генерировать?

- Примитивные типы:

```
st.just(x)              ==>  x, x, x
st.none()               ==>  None, None, None
st.one_of(a, b, c)      ==>  a, a, b, c, a
st.booleans()           ==>  True, False, True
st.integers()           ==>  1, -10, 2, 42
st.floats()             ==>  math.pi, 42.42
```

- Строки и байты:

```
st.text()               ==>  "abra", "cadabra"
st.binary()             ==>  b"\xff\xef", b"ascii"
```

- Коллекции:

```
st.sampled_from(iterable)
st.tuples(st.foo(), st.bar(), st.boo())
st.lists(st.foo())
st.sets(st.foo())
st.dictionaries(st.foo(), st.bar())
```

```
from itertools import chain, repeat, tee

import hypothesis.strategies as st
from hypothesis import given

iterables = st.one_of(st.tuples(st.integers(0, 10)),
                      st.lists(st.integers(0, 10)),
                      st.text().map(iter))

@given(iterables)
def test_rle(it):
    def encode_decode(it):
        return chain.from_iterable(
            repeat(item, count) for item, count in rle(it))

    it, copy = tee(it)
    expected = list(copy)
    assert list(encode_decode(it)) == expected
```

Пакет `hypothesis`: резюме

- Пакет `hypothesis` позволяет удобно формулировать и проверять свойства про Python код.
- Мы обсудили, что можно генерировать, и рассмотрели несколько примеров.
- Львиная часть функциональности `hypothesis` осталась за кадром:
  - как сузить пространство поиска контпримеров?
  - как написать свой генератор?
  - как формулировать зависимые гипотезы?