

Использование unittest

Инструментарий библиотеки:

1. Test case — тестовый случай, базовая единица тестирования.
2. Test fixture — среда исполнения теста. Включает подготовку к тестированию и последующее обнуление данных, используемых в тестовом случае.
3. Test suite — набор тестовых случаев.
4. Test runner — группа запуска тестов. Это множество классов, связанных с запуском и представлением тестов.

Тестовый случай (test case)

Для начала самое главное — научиться создавать тестовые случаи ("тест-кейсы"):

```
class MyTest(unittest.TestCase):  
    def test_usage(self):  
        self.assertEqual(2+2, 4)
```

Обратите внимание, что:

1. класс тестового случая — **потомок** `unittest.TestCase` ;
2. тестирующий метод начинается со слова «test»;
3. для проверки утверждения **используется метод** `self.assertEqual()`.

Если не соблюдать эти правила, то ваш метод либо не будет выполнен, либо ошибка не будет корректно обработана.

Пример тестового случая

Оформим тестирование сортировки методом пузырька в тестовый случай unittest. При этом воспользуемся сравнением `assertCountEqual(a, b)` и проверкой упорядоченности списка `a` за один проход по нему.

```
import unittest  
  
def sort_algorithm(A: list):  
    pass # FIXME  
  
def is_not_in_descending_order(a):  
    """  
    Check if the list a is not descending (means "rather ascending")  
    """  
  
    for i in range(len(a)-1):
```

```

        if a[i] > a[i+1]:
            return False
        return True

class TestSort(unittest.TestCase):
    def test_simple_cases(self):
        cases = ([1], [], [1, 2], [1, 2, 3, 4, 5],
                  [4, 2, 5, 1, 3], [5, 4, 4, 5, 5],
                  list(range(10)), list(range(10, 0, -1)))
        for b in cases:
            a = list(b)
            sort_algorithm(a)
            self.assertEqual(a, b)
            self.assertTrue(is_not_in_descending_order(a))

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

```

Запуск данного теста конечно покажет нам ошибку, но не будет ясно, при каком конкретно случае она случилась.

Можете написать свою версию сортировки и посмотреть на результаты тестирования. Обратите внимание на **ВЫЗОВ** `unittest.main()`, которая запускает все тесты из данного модуля.

Варианты методов assert

Вариант assert	Что проверяет
<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) is True</code>
<code>assertFalse(x)</code>	<code>bool(x) is False</code>
<code>assertIs(a, b)</code>	<code>a is b</code>
<code>assertIsNot(a, b)</code>	<code>a is not b</code>
<code>assertIsNone(x)</code>	<code>x is None</code>
<code>assertIsNotNone(x)</code>	<code>x is not None</code>
<code>assertIn(a, b)</code>	<code>a in b</code>
<code>assertNotIn(a, b)</code>	<code>a not in b</code>
<code>assertIsInstance(a, b)</code>	<code>isinstance(a, b)</code>
<code>assertNotIsInstance(a, b)</code>	<code>not isinstance(a, b)</code>

Заметим, что `assertEqual(a, b)` для строк, последовательностей, списков, кортежей, множеств и словарей осуществляет [специализированную по типу проверку](#).

Есть также проверки, проводящие сравнение и проверки включения:

Вариант assert	Что проверяет
<code>assertAlmostEqual(a, b)</code>	<code>round(a-b, 7) == 0</code>
<code>assertNotAlmostEqual(a, b)</code>	<code>round(a-b, 7) != 0</code>
<code>assertGreater(a, b)</code>	<code>a > b</code>
<code>assertGreaterEqual(a, b)</code>	<code>a >= b</code>
<code>assertLess(a, b)</code>	<code>a < b</code>
<code>assertLessEqual(a, b)</code>	<code>a <= b</code>
<code>assertRegex(s, r)</code>	<code>r.search(s)</code>
<code>assertNotRegex(s, r)</code>	<code>not r.search(s)</code>
<code>assertCountEqual(a, b)</code>	контейнеры равны с точностью до порядка элементов

Выделение подслучая

Для выделения конкретной ситуации, в рамках которой произошла ошибка, удобно использовать метод `self.subTest()`:

```
class TestSort(unittest.TestCase):
    def test_simple_cases(self):
        cases = ([1], [], [1, 2], [1, 2, 3, 4, 5],
                 [4, 2, 5, 1, 3], [5, 4, 4, 5, 5],
                 list(range(1, 10)), list(range(9, 0, -1)))
        for b in cases:
            with self.subTest(case=b):
                a = list(b)
                sort_algorithm(a)
                self.assertEqual(a, b)
                self.assertTrue(is_not_in_descending_order(a))
```

В этом случае тестирование **не остановится на первой же ошибке** в рамках метода `test_simple_cases()`, но продолжится для других случаев. А содержание ошибки из-за передачи параметра `(case=b)` становится информативнее:

```
=====
FAIL: test_simple_cases (__main__.TestSort) (case=[4, 2, 5, 1, 3])
-----
Traceback (most recent call last):
```

```

File "unittest_sort_n2_2.py", line 35, in test_simple_cases
    self.assertTrue(is_not_in_descending_order(a))
AssertionError: False is not true

=====
FAIL: test_simple_cases (__main__.TestSort) (case=[5, 4, 4, 5, 5])
-----

Traceback (most recent call last):
  File "unittest_sort_n2_2.py", line 35, in test_simple_cases
    self.assertTrue(is_not_in_descending_order(a))
AssertionError: False is not true

=====
FAIL: test_simple_cases (__main__.TestSort) (case=[9, 8, 7, 6, 5, 4, 3, 2, 1])
-----

Traceback (most recent call last):
  File "unittest_sort_n2_2.py", line 35, in test_simple_cases
    self.assertTrue(is_not_in_descending_order(a))
AssertionError: False is not true

-----

Ran 1 test in 0.001s

FAILED (failures=3)

```

Для повышения информативности отчёта можно также использовать именованный параметр `msg` этих методов:

```

self.assertEqual(a, b, msg="Elements changed. a = "+str(a))
self.assertTrue(is_not_in_descending_order(a),
                msg="List not sorted. a = "+str(a))

```

Тестирование возбуждения исключений

Хороший программный код должен быть устойчивым в тех случаях, когда его используют с некорректными параметрами. В частности, метод или функция должны возбуждать определённое исключение, когда возникает конкретная внештатная ситуация.

```

self.assertRaises(ValueError, math.sqrt, -1)

```

Обратите внимание, что при использовании метода `assertRaises` **нельзя** вызывать функцию. Мы передаём ему ссылку на функцию и её параметры, чтобы она была вызвана уже внутри метода `assertRaises`, описанного в библиотеке `unittest`.

Если тестируемая функция **не вызывает** ожидаемого исключения, это считается ошибкой:

```

import unittest

```

```
def fib(n):
    return n if n <= 1 else fib(n-1) + fib(n-2)


class ExceptionTest(unittest.TestCase):
    def test_raises(self):
        self.assertRaises(ValueError, fib, -1)


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

Вывод теста:

```
FAIL: test_raises (__main__.ExceptionTest)
-----
Traceback (most recent call last):
  File "C:/Users/tkhirianov/.spyder-py3/temp.py", line 18, in test_raises
    self.assertRaises(ValueError, fib, -1)
AssertionError: ValueError not raised by fib
```

Среда исполнения теста

Для проведения теста нужно создание определённых тестовых условий, определённого состояния **среды исполнения теста** (Test fixture). Например, нужно создать и заполнить определённым образом базу данных, необходимую для проведения операций, подвергающихся проверке. Или же проводится тестирование некоего класса А, использующего объект класса В, который использует объект класса С. В этом случае требуется создать и инициализировать эти объекты.

Базовые правила тестирования:

1. Работа теста **не должна зависеть** от результатов работы других тестов.
2. Тест должен использовать данные, **специально** для него подготовленные, и никакие другие.

Поскольку предыдущие тесты могут повлиять на среду исполнения, её нужно уничтожать и создавать заново для каждого тестового случая. Для этого используются автоматически вызываемые методы

setUp() и tearDown():

```
class TestSort(unittest.TestCase):  
    def setUp(self):  
        self.cases = ([1], [], [1, 2], [1, 2, 3, 4, 5],  
                       [4, 2, 5, 1, 3], [5, 4, 4, 5, 5],  
                       list(range(1, 10)), list(range(9, 0, -1)))  
  
    def test_simple_cases(self):  
        for b in self.cases:  
            with self.subTest(case=b):  
                a = list(b)  
                sort_algorithm(a)  
                self.assertCountEqual(a, b,  
                                     msg=f"Elements changed. a = {str(a)}")
```

```

        self.assertTrue(is_not_in_descending_order(a),
                        msg="List not sorted. a = "+str(a))

def tearDown(self):
    self.cases = None

```



Также существуют методы инициализации среды исполнения для класса (`setUpClass` и `tearDownClass`) и модуля (`setUpModule` и `tearDownModule`), но их неаккуратное использование может привести к нарушению базовых правил, упомянутых выше.

Группировка тестов, управление запуском тестов и интерпретация результатов тестирования

Библиотека unittest также содержит:

- класс `TestSuite`, позволяющий группировать тесты;
- класс `TextTestRunner`, позволяющих запускать группы тестов;
- класс `TestLoader`, управляющий автоматическим созданием объектов `TestSuite`;
- класс `TestResult` для автоматизации анализа результатов тестирования.

В следующем примере используется группировка тестов и их запуск при помощи `TestRunner`:

```

import unittest
import sys

def is_not_in_descending_order(a):
    """
    Check if the list a is not descending (means "rather ascending")
    """
    for i in range(len(a)-1):
        if a[i] > a[i+1]:
            return False
    return True

class TestSort(unittest.TestCase):
    def test_simple_cases(self):
        self.cases = ([1], [], [1, 2], [1, 2, 3, 4, 5],
                      [4, 2, 5, 1, 3], [5, 4, 4, 5, 5],
                      list(range(1, 10)), list(range(9, 0, -1)))
        for b in self.cases:
            with self.subTest(case=b):
                a = list(b)
                sort_algorithm(a)
                self.assertEqual(a, b,
                                msg="Elements changed. a = "+str(a))
                self.assertTrue(is_not_in_descending_order(a),
                                msg="List not sorted. a = "+str(a))

    def test_stability(self):

```

```

self.cases = ([[0] for i in range(5)],
               [[1, 2], [2, 2], [2, 3], [2, 2], [2, 3], [1, 2]],
               [[5, 2], [10, 5], [5, 2], [10, 5], [5, 2], [10, 5]])

for b in self.cases:
    with self.subTest(case=b):
        a = list(b)
        sort_algorithm(a)
        b.sort() # here we are cheating: standard sort is stable
                 # to test stability we will check a[i] is b[i]
        self.assertTrue(all(x is y for x, y in zip(a, b)))

def test_universality(self):
    self.cases = ([4, 2, 8], list('abcdefg'),
                  [True, False],
                  [float(i)/10 for i in range(10, 0, -1)],
                  [[1, 2], [2], [3, 4], [3, 4, 5], [6, 7]])
    for b in self.cases:
        with self.subTest(case=b):
            a = list(b)
            sort_algorithm(a)
            self.assertEqual(a, b,
                             msg="Elements changed. a = "+str(a))
            self.assertTrue(is_not_in_descending_order(a),
                             msg="List not sorted. a = "+str(a))

def bubble_sort(A: list):
    """
    Sorting of list in place. Using Bubble Sort algorithm.
    """
    N = len(A)
    list_is_sorted = False
    bypass = 1
    while not list_is_sorted:
        list_is_sorted = True
        for k in range(N - bypass):
            if A[k] > A[k+1]:
                A[k], A[k+1] = A[k+1], A[k]
                list_is_sorted = False
        bypass += 1

def doing_nothing(A: list):
    """
    Doing nothing with the list A.
    """
    pass

def sort_test_suite():
    suite = unittest.TestSuite()

    suite.addTest(TestSort('test_simple_cases'))

```

```
suite.addTest(TestSort('test_stability'))
suite.addTest(TestSort('test_universality'))
return suite

if True: # __name__ == '__main__':
    runner = unittest.TextTestRunner(stream=sys.stdout, verbosity=2)

    for algo in doing_nothing, bubble_sort:
        print('Testing function ', algo.__doc__.strip())
        test_suite = sort_test_suite()
        sort_algorithm = algo
        runner.run(test_suite)
```

Запустите данный код и посмотрите на результаты тестирования.