**unittest**

**Документация**

* EFACTOR <https://docs.python.org/3/library/unittest.html> - официальная документация
* DEA! почти полностью потянуто с <http://devpractice.ru/python-unittest-lessons/> и пусть будет стыдно тем, кто закрывает в учебных классах интернет.
* <http://www.drdobbs.com/testing/unit-testing-with-python/240165163>

**Плюсы unittest**

Почему unittest? Потому что

* он почти такой же в Java (Junit), C# (Nunit).
* его тесты запускает nose и pytest
* уже стоит (входит в стандартную поставку).

**Что будем тестировать**

Тестируем модуль calc.py

def add(a, b):

return a + b

def sub(a, b):

return a-b

def mul(a, b):

return a \* b

def div(a, b):

return a / b

Напишем тесты, используя unittest и сохраним в файле test\_calc.py

import unittest

import calc

class CalcTest(unittest.TestCase):

def test\_add(self):

self.assertEqual(calc.add(1, 2), 3)

def test\_sub(self):

self.assertEqual(calc.sub(4, 2), 2)

def test\_mul(self):

self.assertEqual(calc.mul(2, 5), 10)

def test\_div(self):

self.assertEqual(calc.div(8, 4), 2)

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

unittest.main()

Запускаем:

python test\_calc.py

Рекомендуем, однако, запускать следующим образом:

python -m unittest test\_calc.py

Так вы легко в командной строке запустите все тесты в файле, все тесты в классе, один тест или вообще все файлы с тестами (см. далее).

Получили:

....

----------------------------------------------------------------------

Ran 4 tests in 0.003s

OK

Лаконично. Запустим с ключом -v (verbose) и получим отчет по каждому тесту.

test\_add (test\_calc.CalcTest) ... ok

test\_div (test\_calc.CalcTest) ... ok

test\_mul (test\_calc.CalcTest) ... ok

test\_sub (test\_calc.CalcTest) ... ok

----------------------------------------------------------------------

Ran 4 tests in 0.004s

OK

Как мы придумали название класса и методов?

*ИмяТестируемойСущности*Tests.*ИмяТестируемойСущности* – это некоторая логическая единица, тесты для которой нужно написать. В нашем случае – это калькулятор, поэтому мы выбрали имя CalcTests[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/CalcTests?topicparent=Cintro.PythonUnittest). Если бы у нашего калькулятора был большой набор поддерживаемых функций, то тестирование простых функций (сложение, вычитание, умножение и деление) можно было бы вынести в отдельный класс и назвать его например так: CalcSimpleActionsTests[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/CalcSimpleActionsTests?topicparent=Cintro.PythonUnittest).

**Терминология**

* **unittest** – это framework для тестирования в Python, который позволяет разрабатывать автономные тесты, собирать тесты в коллекции, обеспечивает независимость тестов от framework’а отчетов и т.д. Основными структурными элемента каркаса unittest являются:
* **Test fixture** (функции и методы, которые запускаются для создания соответствующего окружения для теста) – обеспечивает подготовку окружения для выполнения тестов, а также организацию мероприятий по их корректному завершению (например очистка ресурсов). Подготовка окружения может включать в себя создание баз данных, запуск необходим серверов и т.п.
  + **setup** - подготовка к тесту
  + **teardown** - зачистка после окончания (например, удаляем часть таблицы с транзакциями позже начала теста)
* **Test case** (1 тест) – это элементарная единица тестирования, в рамках которой проверяется работа компонента тестируемой программы (метод, класс, поведение и т.п.). Для реализации этой сущности используется класс TestCase[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/TestCase?topicparent=Cintro.PythonUnittest).
* **Test suite** (набор тестов, может включать не только тесты, но и другие наборы) – это коллекция тестов, которая может в себя включать как отдельные test case’ы так и целые коллекции (т.е. можно создавать коллекции коллекций). Коллекции используются с целью объединения тестов для совместного запуска.
* **Test runner** ("пускатель" тестов) – это компонент, который координирует взаимодействие запуска тестов и предоставляет пользователю результат их выполнения. Test runner может иметь графический интерфейс, текстовый интерфейс или возвращать какое-то заранее заданное значение, которое будет описывать результат прохождения тестов (в рамках билда очередной версии продукта).

Вся работа по написанию тестов заключается в том, что мы разрабатываем отдельные тесты в рамках test case’ов, собираем их в модули и запускаем, если нужно объединить несколько test case’ов, для их совместного запуска, они помещаются в test suite’ы, которые помимо test case’ов могут содержать другие test suite’ы.

**Запуск**

**Из командной строки (CLI - command line interface)**

EFACTOR не забываем ключ **-v** (verbose)

Всего файла

python -m unittest test\_calc.py

Одного класса из файла

python -m unittest test\_calc.CalcTest

Одного теста

python -m unittest test\_calc.CalcTest.test\_sub

EFACTOR Запуск всех тестов, что найдет TestDiscovery[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/TestDiscovery?topicparent=Cintro.PythonUnittest)

python -m unittest

python -m unittest discover

Эти формы запуска эквивалентны.

По умолчанию из текущей директории находятся все файлы с маской test\*.py и запускаются.

Если нужно запустить из другой директории или с другой маской, используйте

python -m unittest discover project\_directory "\*\_test.py"

**GUI**

Иногда проще поставить GUI, чем обучать людей. *Один* из вариантов - поставить [Cricket](https://github.com/pybee/cricket)

Установка:  
pip install cricket

Запуск:   
cricket-unittest

Все разнообразие вариантов: <https://wiki.python.org/moin/PythonTestingToolsTaxonomy#GUI_Testing_Tools>

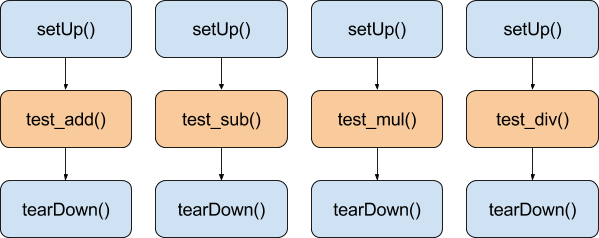
**Методы**

Для того, чтобы метод класса выполнялся как тест, необходимо, чтобы он начинался со слова test. Несмотря на то, что методы framework’а unittest написаны не в соответствии с PEP 8 (ввиду того, что идейно он наследник xUnit), мы все же рекомендуем следовать правилам стиля для Python везде, где это возможно. Поэтому имена тестов будем начинать с префикса test\_. Далее, под словом тест будем понимать метод класса-наследника от TestCase[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/TestCase?topicparent=Cintro.PythonUnittest), который начинается с префикса test\_.

Методы класса unittest.TestCase:

* методы, используемые при запуске тестов;
* методы, используемые при непосредственном написании тестов (проверка условий, сообщение об ошибках);
* методы, позволяющие собирать информацию о самом тесте.

**Методы, используемые при запуске тестов (setUp, tearDown)**



* **setUp** - вызывается перед запуском теста (т.е уровень метода)
* **tearDown** - вызывается после окончания теста (уровень метода)
* **setUpClass** - вызывается перед запуском всех тестов класса (уровень класса), требует декоратора @classmethod
* **tearDownClass** - вызывается после окончания всех тестов класса (уровень класса), требует декоратора @classmethod
* **skipTest(reason)** - для пропуска данного теста

@classmethod

def setUpClass(cls):

# что нужно сделать перед всеми тестами этого класса

@classmethod

def tearDownClass(cls):

# что нужно сделать после окончания всех тестов этого класса

По умолчанию все эти методы ничего не делают.

**Assert'ы**

**fail(msg = None)** - в тесте произошла ошибка.

Проверка и генерация ошибок:

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

Сравнения и поиск:

|  |  |
| --- | --- |
| [assertAlmostEqual(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertAlmostEqual) | round(a-b, 7) == 0 |
| [assertNotAlmostEqual(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertNotAlmostEqual) | round(a-b, 7) != 0 |
| [assertGreater(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertGreater) | a > b |
| [assertGreaterEqual(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertGreaterEqual) | a >= b |
| [assertLess(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertLess) | a < b |
| [assertLessEqual(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertLessEqual) | a <= b |
| [assertRegex(s, r)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertRegex) | r.search(s) |
| [assertNotRegex(s, r)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertNotRegex) | not r.search(s) |
| [assertCountEqual(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertCountEqual) | a и b имеют одинаковые элементы (порядок неважен) |

Типо-зависимые assert’ы, которые используются при вызове assertEqual(). Приводятся на тот случай, если необходимо использовать конкретный метод.

|  |  |
| --- | --- |
| [assertMultiLineEqual(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertMultiLineEqual) | строки (strings) |
| [assertSequenceEqual(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertSequenceEqual) | последовательности (sequences) |
| [assertListEqual(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertListEqual) | списки (lists) |
| [assertTupleEqual(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertTupleEqual) | кортежи (tuplse) |
| [assertSetEqual(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertSetEqual) | множества или неизменяемые множества (frozensets) |
| [assertDictEqual(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertDictEqual) | словари (dicts) |

Контроль выбрасываемых исключений и warning'ов (да, их тоже нужно проверить!)

|  |  |
| --- | --- |
| [assertRaises(exc, fun, \*args, \*\*kwds)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertRaises) | Функция fun(\*args, \*\*kwds) вызывает исключение exc |
| [assertRaisesRegex(exc, r, fun, \*args, \*\*kwds)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertRaisesRegex) | Функция fun(\*args, \*\*kwds) вызывает исключение exc, сообщение которого совпадает с регулярным выражением r |
| [assertWarns(warn, fun, \*args, \*\*kwds)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertWarns) | Функция fun(\*args, \*\*kwds) выдает сообщение warn |
| [assertWarnsRegex(warn, r, fun, \*args, \*\*kwds)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertWarnsRegex) | Функция fun(\*args, \*\*kwds) выдает сообщение warn и оно совпадает с регулярным выражением r |

**Методы, позволяющие собирать информацию о самом тесте**

* **countTestCases()** - возвращает количество тестов у наследника от TestCase[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/TestCase?topicparent=Cintro.PythonUnittest).
* **id()** - Возвращает строковый идентификатор теста. Как правило это полное имя метода, включающее имя модуля и имя класса.
* **shortDescription()** - Возвращает описание теста, которое представляет собой первую строку docstring’а метода, если его нет, то возвращает None.

**Пример использования методов**

Добавим методы в test\_calc.py

import unittest

import calc

class CalcTest(unittest.TestCase):

"""Calc tests"""

@classmethod

def setUpClass(cls):

"""Set up for class"""

print("setUpClass")

print("==========")

@classmethod

def tearDownClass(cls):

"""Tear down for class"""

print("==========")

print("tearDownClass")

def setUp(self):

"""Set up for test"""

print("Set up for [" + self.shortDescription() + "]")

def tearDown(self):

"""Tear down for test"""

print("Tear down for [" + self.shortDescription() + "]")

print("")

def test\_add(self):

"""Add operation test"""

print("id: " + self.id())

self.assertEqual(calc.add(1, 2), 3)

def test\_sub(self):

"""Sub operation test"""

print("id: " + self.id())

self.assertEqual(calc.sub(4, 2), 2)

def test\_mul(self):

"""Mul operation test"""

print("id: " + self.id())

self.assertEqual(calc.mul(2, 5), 10)

def test\_div(self):

"""Div operation test"""

print("id: " + self.id())

self.assertEqual(calc.div(8, 4), 2)

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

unittest.main()

При запуске python -m unittest -v test\_calc.py получим

setUpClass

==========

test\_add (simple\_ex.CalcTest)

Add operation test ... Set up for [Add operation test]

id: simple\_ex.CalcTest.test\_add

Tear down for [Add operation test]

ok

test\_div (simple\_ex.CalcTest)

Div operation test ... Set up for [Div operation test]

id: simple\_ex.CalcTest.test\_div

Tear down for [Div operation test]

ok

test\_mul (simple\_ex.CalcTest)

Mul operation test ... Set up for [Mul operation test]

id: simple\_ex.CalcTest.test\_mul

Tear down for [Mul operation test]

ok

test\_sub (simple\_ex.CalcTest)

Sub operation test ... Set up for [Sub operation test]

id: simple\_ex.CalcTest.test\_sub

Tear down for [Sub operation test]

ok

==========

tearDownClass

----------------------------------------------------------------------

Ran 4 tests in 0.016s

OK

* setup метод
* методы тестов в **алфавитном** порядке (а если я хочу определенный порядок? от простых к сложным)
* teardown

**Класс TestSuite**[**?**](http://acm.mipt.ru/twiki/bin/edit/Cintro/TestSuite?topicparent=Cintro.PythonUnittest)

* объединить тесты в наборы тестов
* интерфейс для запуска тестов TestRunner[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/TestRunner?topicparent=Cintro.PythonUnittest)'ом
* **addTest(test)** - Добавляет TestCase[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/TestCase?topicparent=Cintro.PythonUnittest) или TestSuite[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/TestSuite?topicparent=Cintro.PythonUnittest) в группу.
* **addTests(tests)** - Добавляет все TestCase[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/TestCase?topicparent=Cintro.PythonUnittest) и TestSuite[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/TestSuite?topicparent=Cintro.PythonUnittest) объекты в группу, итеративно проходя по элементам переменной tests.
* **run(result)** - Запускает тесты из данной группы.
* **countTestCases()** - Возвращает количество тестов в данной группе (включает в себя как отдельные тесты, так и подгруппы).

Вспомним calc.py и test\_calc.py.

Напишем пускалку тестов - файл test\_runner.py

import unittest

import calc\_tests

calcTestSuite = unittest.TestSuite()

calcTestSuite.addTest(unittest.makeSuite(calc\_tests.CalcTest))

runner = unittest.TextTestRunner(verbosity=2)

runner.run(calcTestSuite)

Запуск:  
  
python test\_runner.py

**Тестирование новой фукнциональности в старом классе**

Допишем в файл calc.py новые функции. Простестируем старые функции (сначала) и потом запустим тесты новых функций (напишем новый класс).

def add(a, b):

return a + b

def sub(a, b):

return a-b

def mul(a, b):

return a \* b

def div(a, b):

return a / b

def sqrt(a): # новая функциональность

return a\*\*0.5

def pow(a, b): # новая функциональность

return a\*\*b

Добавим тесты для новых функций, создав новый класс с именем CalcExTests[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/CalcExTests?topicparent=Cintro.PythonUnittest) (расширенные функции калькулятора) с тестами для sqrt() и pow(), а класс CalcTest[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/CalcTest?topicparent=Cintro.PythonUnittest) переименуем в CalcBasicTests[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/CalcBasicTests?topicparent=Cintro.PythonUnittest) (базовые функции калькулятора).

Модуль test\_calc.py

import unittest

import calc

class CalcBasicTests(unittest.TestCase):

def test\_add(self):

self.assertEqual(calc.add(1, 2), 3)

def test\_sub(self):

self.assertEqual(calc.sub(4, 2), 2)

def test\_mul(self):

self.assertEqual(calc.mul(2, 5), 10)

def test\_div(self):

self.assertEqual(calc.div(8, 4), 2)

class CalcExTests(unittest.TestCase):

def test\_sqrt(self):

self.assertEqual(calc.sqrt(4), 2)

def test\_pow(self):

self.assertEqual(calc.pow(3, 3), 27)

Установим в test\_runner порядок запуска наборов тестов:

import unittest

import calc\_tests

calcTestSuite = unittest.TestSuite()

calcTestSuite.addTest(unittest.makeSuite(calc\_tests.CalcBasicTests))

calcTestSuite.addTest(unittest.makeSuite(calc\_tests.CalcExTests))

print("count of tests: " + str(calcTestSuite.countTestCases()) + "\n")

runner = unittest.TextTestRunner(verbosity=2)

runner.run(calcTestSuite)

Запускаем python test\_runner.py

count of tests: 6

test\_add (calc\_tests.CalcBasicTests) ... ok

test\_div (calc\_tests.CalcBasicTests) ... ok

test\_mul (calc\_tests.CalcBasicTests) ... ok

test\_sub (calc\_tests.CalcBasicTests) ... ok

test\_pow (calc\_tests.CalcExTests) ... ok

test\_sqrt (calc\_tests.CalcExTests) ... ok

----------------------------------------------------------------------

Ran 6 tests in 0.000s

OK

Сначала выполняются тесты из класса CalcBasicTests[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/CalcBasicTests?topicparent=Cintro.PythonUnittest), потом из CalcExTests[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/CalcExTests?topicparent=Cintro.PythonUnittest).

**Загрузка и запуск тестов**

**Класс TestLoader**[**?**](http://acm.mipt.ru/twiki/bin/edit/Cintro/TestLoader?topicparent=Cintro.PythonUnittest)

**TestLoader** - используется для создания групп из классов и модулей

Модуль test\_runner.py

import unittest

import calc\_tests

testCases = []

testCases.append(calc\_tests.CalcBasicTests)

testCases.append(calc\_tests.CalcExTests)

testLoad = unittest.TestLoader()

suites = []

for tc in testCases:

suites.append(testLoad.loadTestsFromTestCase(tc))

res\_suite = unittest.TestSuite(suites )

runner = unittest.TextTestRunner(verbosity=2)

runner.run(res\_suite)

Запустим.

$ python3 test\_runner.py

test\_add (calc\_tests.CalcBasicTests) ... ok

test\_div (calc\_tests.CalcBasicTests) ... ok

test\_mul (calc\_tests.CalcBasicTests) ... ok

test\_sub (calc\_tests.CalcBasicTests) ... ok

test\_pow (calc\_tests.CalcExTests) ... ok

test\_sqrt (calc\_tests.CalcExTests) ... ok

----------------------------------------------------------------------

Ran 6 tests in 0.007s

OK

* **loadTestsFromTestCase(testCaseClass)** - возвращает группу со всеми тестами из класса testCaseClass.  
  Напоминаем, что под тестом понимается модуль, начинающийся со слова test. Используя этот метод, можно создать список групп тестов, где каждая группа создается на базе классов-наследников от TestCase[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/TestCase?topicparent=Cintro.PythonUnittest), объединенных предварительно в список.
* **loadTestsFromModule(module, pattern=None)** - Загружает все тесты из модуля module. Если модуль поддерживает load\_tests протокол, то будет вызвана соответствующая функция модуля и ей будет передан в качестве аргумента (третьим по счету) параметр pattern.
* **loadTestsFromName(name, module=None)** - Загружает тесты в соответствии с параметром name. Параметр name – это имя, разделенное точками. С помощью этого имени указывается уровень, начиная с которого будут добавляться тесты.
* **getTestCaseNames(testCaseClass)** - Возвращает список имен методов-тестов из класса testCaseClass.

Еще пример файла test\_runner.py:

import unittest

import calc\_tests

testLoad = unittest.TestLoader()

suites = testLoad.loadTestsFromModule(calc\_tests)

runner = unittest.TextTestRunner(verbosity=2)

runner.run(suites)

Запустим и получим:

test\_add (calc\_tests.CalcBasicTests) ... ok

test\_div (calc\_tests.CalcBasicTests) ... ok

test\_mul (calc\_tests.CalcBasicTests) ... ok

test\_sub (calc\_tests.CalcBasicTests) ... ok

test\_pow (calc\_tests.CalcExTests) ... ok

test\_sqrt (calc\_tests.CalcExTests) ... ok

----------------------------------------------------------------------

Ran 6 tests in 0.016s

OK

Если в модуле test\_runner.py заменить строку

**suites = testLoad.loadTestsFromModule(calc\_tests)**

на

**suites = testLoad.loadTestsFromName('calc\_tests.CalcBasicTests')**

то будут выполнены только тесты из класса CalcBasicTests[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/CalcBasicTests?topicparent=Cintro.PythonUnittest).

test\_add (calc\_tests.CalcBasicTests) ... ok

test\_div (calc\_tests.CalcBasicTests) ... ok

test\_mul (calc\_tests.CalcBasicTests) ... ok

test\_sub (calc\_tests.CalcBasicTests) ... ok

----------------------------------------------------------------------

Ran 4 tests in 0.002s

OK

DEA! Можно сделать разные наборы - для полного тестирования (например, на несколько часов или дней) и для быстрой проверки после каждой сборки.

**Класс TestResult**[**?**](http://acm.mipt.ru/twiki/bin/edit/Cintro/TestResult?topicparent=Cintro.PythonUnittest)

Класс TestResult[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/TestResult?topicparent=Cintro.PythonUnittest) используется для сбора информации о результатах прохождения тестов.

Модифицируем файл test\_runner.py, чтобы печатался только итог, но развернутый итог.

import unittest

import calc\_tests

testLoad = unittest.TestLoader()

suites = testLoad.loadTestsFromModule(calc\_tests)

testResult = unittest.TestResult()

runner = unittest.TextTestRunner(verbosity=1) # понизим говорливость тестов

testResult = runner.run(suites) # станем анализировать, что вернул run

print("errors")

print(len(testResult.errors))

print("failures")

print(len(testResult.failures))

print("skipped")

print(len(testResult.skipped))

print("testsRun")

print(testResult.testsRun)

Запустим и получим:

......

----------------------------------------------------------------------

Ran 6 tests in 0.001s

OK

errors

0

failures

0

skipped

0

testsRun

6

Получили расширенное summary.

**Класс TextTestRunner**[**?**](http://acm.mipt.ru/twiki/bin/edit/Cintro/TextTestRunner?topicparent=Cintro.PythonUnittest)

Объекты класса TextTestRunner[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/TextTestRunner?topicparent=Cintro.PythonUnittest) используются для запуска тестов. Среди параметров, которые передаются конструктору класса, можно выделить verbosity, по умолчанию он равен 1, если создать объект с verbosity=2, то будем получать расширенную информацию о результатах прохождения тестов. Для запуска тестов используется метод run(), которому в качестве аргумента передается класс-наследник от TestCase[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/TestCase?topicparent=Cintro.PythonUnittest) или группа (TestSuite[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/TestSuite?topicparent=Cintro.PythonUnittest)).

В наших примерах TextTestRunner[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/TextTestRunner?topicparent=Cintro.PythonUnittest) используется в модуле test\_runner.py в строчках:

runner = unittest.TextTestRunner(verbosity=2)

testResult = runner.run(suites)

В первой строке создается объект класса TextTestRunner[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/TextTestRunner?topicparent=Cintro.PythonUnittest) с verbosity=2, а во второй строке запускаются тесты из группы suites, результат тестирования попадает в объект testResult, атрибуты которого можно анализировать в дальнейшем.

**Пропуск тестов и классов**

Зачем пропускать тесты? Например, у нас известный баг, который поломал нам часть тестов. Мы ведем разработку над другой частью, которая независима и не хотим вчитываться в результаты запуска тестов - это наши тесты сломались или известные чужие (если НЕ известные чужие, то мы вчитываемся разбираемся с поломками новых тестов).

Возьмем простой test\_runner.py, который запускает все тесты из всех классов.

import unittest

import calc\_tests

calcTestSuite = unittest.TestSuite()

calcTestSuite.addTest(unittest.makeSuite(calc\_tests.CalcBasicTests))

calcTestSuite.addTest(unittest.makeSuite(calc\_tests.CalcExTests))

runner = unittest.TextTestRunner(verbosity=2)

runner.run(calcTestSuite)

Для пропуска тестов можно закоментаривать часть тестов (нет об этом диагностики, долго). Но хотим быстрый способ НЕ делать часть тестов или классов тестов.

**Пропуск тестов**

Добавьте к тесту декоратор

@unittest.skip(reason)

Например:

class CalcBasicTests(unittest.TestCase):

@unittest.skip("Temporaly skip test\_add")

def test\_add(self):

self.assertEqual(calc.add(1, 2), 3)

def test\_sub(self):

self.assertEqual(calc.sub(4, 2), 2)

def test\_mul(self):

self.assertEqual(calc.mul(2, 5), 10)

def test\_div(self):

self.assertEqual(calc.div(8, 4), 2)

Запустим:

test\_add (calc\_tests.CalcBasicTests) ... skipped 'Temporarily skipped'

test\_div (calc\_tests.CalcBasicTests) ... ok

test\_mul (calc\_tests.CalcBasicTests) ... ok

test\_sub (calc\_tests.CalcBasicTests) ... ok

test\_pow (calc\_tests.CalcExTests) ... ok

test\_sqrt (calc\_tests.CalcExTests) ... ok

----------------------------------------------------------------------

Ran 6 tests in 0.003s

OK (skipped=1)

Заметьте, у нас есть информация, что все остальные тесты прошли хорошо, и что у нас есть специально пропущенные тесты (которые нужно будет в конце концов выполнить).

**Условный пропуск тестов**

Добавьте декоратор

@unittest.skipIf(condition, reason)

Тест будет пропущен, если условие (condition) истинно.

@unittest.skipUnless(condition, reason)

Тест будет пропущен если, условие (condition) не истинно.

Зачем это нужно?

Условный пропуск тестов можно использовать в ситуациях, когда те или иные тесты зависят от версии программы, например: в новой версии уже не поддерживается часть методов; или тесты могут быть платформозависимые, например: ряд тестов могут выполняться только под операционной системой MS Windows. Условие записывается в параметр condition, текстовое описание – в reason.

**Пропуск классов**

Используйте декоратор

@unittest.skip(reason)

который записывается перед объявлением класса. В результате все тесты из данного класса не будут выполнены. В рамках нашего примера с математическими действиями, для исключения из процесса тестирования методов sqrt и pow поместим декоратор skip перед объявлением класса CalcExTests[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/CalcExTests?topicparent=Cintro.PythonUnittest).

import unittest

import calc

class CalcBasicTests(unittest.TestCase):

def test\_add(self):

self.assertEqual(calc.add(1, 2), 3)

def test\_sub(self):

self.assertEqual(calc.sub(4, 2), 2)

def test\_mul(self):

self.assertEqual(calc.mul(2, 5), 10)

def test\_div(self):

self.assertEqual(calc.div(8, 4), 2)

@unittest.skip("Skip CalcExTests")

class CalcExTests(unittest.TestCase):

def test\_sqrt(self):

self.assertEqual(calc.sqrt(4), 2)

def test\_pow(self):

self.assertEqual(calc.pow(3, 3), 27)

Запустим все тесты test\_runner.py и получим:

test\_add (calc\_tests.CalcBasicTests) ... ok

test\_div (calc\_tests.CalcBasicTests) ... ok

test\_mul (calc\_tests.CalcBasicTests) ... ok

test\_sub (calc\_tests.CalcBasicTests) ... ok

test\_pow (calc\_tests.CalcExTests) ... skipped 'Skip CalcExTests'

test\_sqrt (calc\_tests.CalcExTests) ... skipped 'Skip CalcExTests'

----------------------------------------------------------------------

Ran 6 tests in 0.001s

OK (skipped=2)

**Задачи**

Написать тесты ко всем классам из [PythonOOPTask1](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonOOPTask1)

-- [TatyanaDerbysheva](http://acm.mipt.ru/twiki/bin/view/Main/TatyanaDerbysheva) - 12 Nov 2017