Python

Семинар 3

Преподаватель: Дмитрий Косицин
BSU FAMCS (Fall'19)

Проверка аргументов

assert – statement, позволяющий проверить на истинность некоторое выражение

```
def calculate_binomial_mean(n, p):
    assert n > 0, 'number of experiments must be positive'
    assert 0 <= p <= 1, 'probability must be in [0; 1]'
    return n * p
```

Важно! Скобки **assert** имеют смысл проверки кортежа на пустоту.

Юнит-тесты

Общая идея юнит-тестов

- Разбить код на независимые части (юниты)
- Тестировать каждую часть отдельно

Преимущества

- Нужно меньше тестов
- Проще отлаживать

Недостатки

- Нужны тесты, проверяющие взаимодействие юнитов

doctest

Библиотека **doctest** (Py2, Py3) позволяет расположить тест непосредственно в документации, чтобы и показать , и проверить, как работает функция.

Недостаток подхода в его сложности: проверка некорректных входных данных или результатов сложных типов трудоемка.

```
def factorial(n):
    """Return the factorial of n, an exact integer >= 0.

    >>> factorial(5)
    120
    """
    pass  # implementation is here

if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

unittest

Тесты можно писать, используя библиотеку **unittest** (Py2, Py3), что позволяет, в частности, группировать тесты в test cases, а также использовать множество удобных проверок.

```
import unittest

class TestFactorial(unittest.TestCase):
    def test_simple(self):
        self.assertEqual(factorial(5), 120)

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

Возможности unittest

- Проверка различных типов и видов возвращаемых значений: ***assertTrue***, ***assertIsNone***, ***assertAlmostEqual***, ***assertRaisesRegexp*** и др.
- Возможность подготовить тестирование: методы ***setup*** (***setUpClass***) и ***teardown*** (***tearDownClass***) – вызываются перед и после каждого запуска теста (класса test case), создавая и удаляя используемые объекты.
- Возможность пропустить тест по некоторому условию (см. unittest.SkipTest).

Также есть библиотека **pytest**, в которой все проверки можно проводить с помощью **assert** statement'ов, и библиотека **nose**, объединяющая все виды тестов.

Подмена объектов

При помощи библиотеки **mock** (в стандартной библиотеке с Python 3.3, *примеры*) можно подменить любой объект, будь то поток ввода-вывода ***stdout***, некоторый модуль, класс, атрибут, свойство, метод.

```
from io import StringIO


class InterfaceTestCase(unittest.TestCase):
    def setUp(self):
        self._stdout_mock = self._setup_stdout_mock()

    def _setup_stdout_mock(self):
        patcher = mock.patch('sys.stdout', new=StringIO())
        patcher.start()
        self.addCleanup(patcher.stop)
        return patcher.new
```

Профилирование

Замер времени исполнения

Для замера времени исполнения используйте модуль **timeit**.

```
>>> import timeit
>>> timeit.timeit('"-".join(str(n) for n in range(10000))',
number=1000)
2.027786269876367 3
>>> timeit.timeit('"-".join(str(n) for n in
list(range(10000)))', number=1000)
2.269286573095144
```

Замечание. Также доступна функция *repeat* (повторять эксперимент несколько раз).

Профилирование

Для профилирования есть модули **cProfile** и **Profile**.

```
>>> import cProfile  # or Profile - pure Python implementation
>>> profiler = cProfile.Profile()
>>> profiler.run_call(calculate_binomial_mean, 10, 0.5)
# another way: run('calculate_binomial_mean(10, 0.5)')
>>> profiler.print_stats()
```

Вызов выведет статистику по времени выполнения функции, в том числе по всем вложенным (если есть).

Библиотеки для профилирования

Библиотеки для профилирования:

- Просмотр времени выполнения каждой строки: line_profiler
- Использование памяти: memory_profiler, pympler и др.
- Визуализация профилирования: SnakeViz
- Прочие инструменты: ссылка
- Сравнение расходов памяти объектами: ссылка

Сравнение расходов памяти

Сравним объекты с полями ‘x’, ‘y’ и ‘z’ со значениями 1, 2, 3:

Тип объекта	Байт в памяти
dict	240
class instance	56
class instance with __slots__	40 + 24
tuple (namedtuple)	40 + 8 + 24
Cython	16 + 12 + 4
Numpy	(см. большие массивы данных)