Белая магия Python

Юревич Юрий
http://pyobject.ru
the.pythy@gmail.com

Конференция по Ruby и Python
Омск, 21 июня 2008
http://rupy.ru/

Python это не Java/C# без компилятора

План

- Интроспекция
- Функции
  - ...
- Последовательности
  - ...

Интроспекция

- Всё - объект
- Информация об объекте в рантайме
    - Документация
    - Атрибуты, методы
    - Идентификация
    - Наследование
    - ...

Интроспекция

```
>>> def introspect():
        local_s = "some string"
        print "Current namespace: %s" % dir()
        print "Locals: %s" % locals()
        print "Globals: %s" % globals()

>>> s = "another string"
>>> n = 15
>>> introspect()
Current namespace: ['local_s']
Locals: {'local_s': 'some_string'}
Globals: {'__builtins__': <module '__builtin__' (built-in)>,
'introspect': <function introspect at 0x851a304>, 'n': 15, 's':
'another_string', '__name__': '__main__', '__doc__': None}
>>> dir()
['__builtins__', '__doc__', 'introspect', 'n', 's']
>>> isinstance(n, int)
True
>>> hasattr(n, 'get')
False
```

Функции

- Функции — объекты
- Значение аргументов по умолчанию
- Переменное число аргументов
- Анонимные функции
- Замыкания
- Декораторы

Функции — объекты

```
>>> def mult(x, y):
        return x * y

>>> moo = mult
>>> moo
<function mult at 0x8516c34>
>>> def executor(a, b, fun):
        return fun(a, b)

>>> executor(2, 3, moo)
6
```

- Функции — это тип данных
- Присваивание
- Передача параметром

Значение аргументов по умолчанию

```
>>> def say(greet='Hello', name='world'):
        print "%s, %s!" % (greet, name)

>>> say()
Hello, world!
>>> say('Hi')
Hi, world!
>>> say('Hi', 'Dorian')
Hi, Dorian!
>>> say(name='people')
Hello, people!
>>> say(name='everybody', greet='Hi')
Hi, everybody!
>>> say('Good day', name='sir')
Good day, sir!
>>> say(extra='Ooops')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: say() got an unexpected keyword argument 'extra'
```

- Аргументы передаются:
  - По позиции (как список)
  - По имени (как словарь) порядок не важен
  - Смешанно (вначале список, потом словарь)

Переменное число аргументов

```
>>> def printargs(f, s=2,
                  *args, **kwargs):
        print "pos (req) arg: %s" % f
        print "keyword arg: %s" % s
        print "*args: %s" % (args,)
        print "**kwargs: %s" % (kwargs,)
>>> printargs(1, 3, 4, 5)
pos (req) arg: 1
keyword arg: 3
*args: (4, 5)
**kwargs: {}
>>> printargs(s=3, f=1, third=4)
pos (req) arg: 1
keyword arg: 3
*args: ()
**kwargs: {'third': 4}
>>> printargs(1, 3, 4, 5,
              extra='foo', key='bar')
pos (req) arg: 1
keyword arg: 3
*args: (4, 5)
**kwargs: {'key': 'bar', 'extra': 'foo'}
```

- *args — позиционные аргументы
- **kwargs — именованные аргументы
- Вначале позиционные, потом именованные

Анонимные функции

```
>>> def executor(a, b, fun):
        return fun(a, b)

>>> def mult(x, y):
        return x * y

>>> summ = lambda x, y: x + y
>>> mult
<function mult at 0x851633c>
>>> summ
<function <lambda> at 0x8516304>
>>> executor(2, 3, mult)
6
>>> executor(2, 3, summ)
5
```

- lambda *args: expr
- Как и с обычной функцией:
  - Присваивать переменной
  - Передавать параметром
  - Возвращать
  - ...

Замыкания

```
>>> def adder(value_add_to):
        def inner(x):
            return x + value_add_to
        return inner

>>> f = adder(2)
>>> f
<function inner at 0x8516614>
>>> f(5)
7
>>> g = adder(-3)
>>> g
<function inner at 0x851648c>
>>> g(5)
2
```

- Замыкания (closure)
- Замыкание — функция в функции
- Внутренняя функция использует внешний контекст

Декораторы

```
>>> def required_int(decorated):
        def wrapper(n):
            if not isinstance(n, int):
                raise TypeError
            return decorated(n)
        return wrapper
>>> f = lambda x: x * 2
>>> f(2)
4
>>> f('x')
'xx'
>>> f = required_int(f)
>>> f
<function wrapper at 0x851672c>
>>> f(2)
4
>>> f('x')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 4, in wrapper
TypeError
```

- Паттерн «из коробки»
- Изменение поведение без изменения тела функции
- «Синтаксический сахар»
  - ```
    def f(x):
        ...
    f = decorator(f)
    ```
  - ```
    @decorator
    def f(x):
        ...
    ```

Декораторы: пример

```
>>> class memoize(object):
        def __init__(self):
            self.memory = {}
        def __call__(self, decorated):
            def wrapper(x, y):
                if (x, y) in self.memory:
                    return self.memory[(x, y)]
                else:
                    res = decorated(x, y)
                    self.memory[(x, y)] = res
                    return res
            return wrapper

>>> @memoize()
    def summ(a, b):
        print "Calculating summ of %s and %s" % (a, b)
        return a + b

>>> summ(1, 2)
Calculating summ of 1 and 2
3
>>> summ(1, 2)
3
```

Последовательности

- Итераторы
- Генераторы
- Списковые выражения (list comprehensive)
- Выражения-генераторы (generator expressions)

Итераторы

```
>>> iterator = iter([1, 2, 3, 4])
>>> iterator.next()
1
>>> for i in iterator: print i

2
3
4
>>> for i in iterator: print i

>>> iterator.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in
<module>
StopIteration
```

- Итератор:
  - Проитерировать
  - Получить следующий элемент
  - По окончанию выкидывает исключение StopIteration

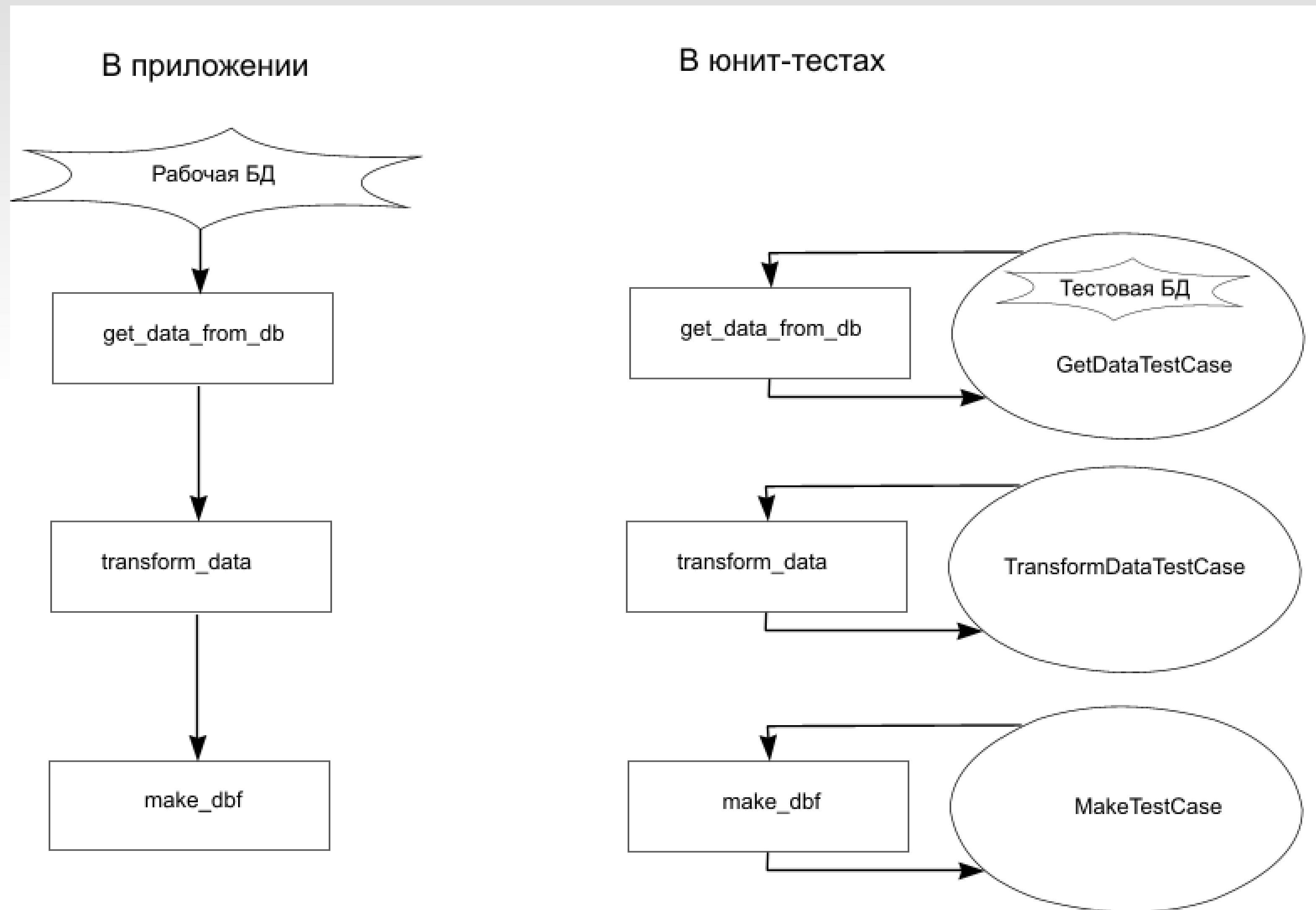
Генераторы

```
>>> def countdown(n):
        print "Counting down from %d" % (n,)
        while n > 0:
            yield n
            n -= 1

>>> x = countdown(5)
>>> x
<generator object at 0x85205cc>
>>> x.next()
Counting down from 5
5
>>> x.next()
4
>>> for i in x: print i
3
2
1
>>> x.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

- Функция, только
  - Не возвращает результат (return — конец последовательности)
  - Генерирует последовательность поэлементно (yield)
  - «Запоминает» состояние между генерациями

Генераторы: pipeline данных



Списковые выражения Выражения-генераторы

```
>>> source = [-1, 2, -5, 4, 3]
>>> res = []
>>> for i in source:
        if i > 0:
            res.append(i**2)
>>> res
[4, 16, 9]
>>> list_compr =
[i**2 for i in source if i > 0]
>>> list_compr
[4, 16, 9]
>>> gen_expr =
(i**2 for i in source if i > 0)
>>> gen_expr
<generator object at 0x85211ec>
>>> gen_expr.next()
4
>>> list(gen_expr)
[16, 9]
```

- Списковые выражения:
  - [<expr> for *(for|if)]
  - Быстрее цикла for
- Выражения-генераторы:
  - (<expr> for *(for|if))
  - Вычисляются по факту

Еще магии?

http://pyobject.ru/p/pymagic

Немного черной магии

```
>>> class A(object):
        def a(self):
            print "I'm an A instance"

>>> class B(object):
        def b(self):
            print "I'm a B instance"
>>> z = A()
>>> z
<__main__.A object at 0x8528dcc>
>>> z.a()
I'm an A instance
>>> isinstance(z, A)
True
>>> # черная магия -- смена класса в рантайме
>>> z.__class__ = B
>>> z
<__main__.B object at 0x8528dcc>
>>> z.b()
I'm a B instance
>>> isinstance(z, B)
True
```

Вопросы?

Пишите письма
mailto:the.pythy@gmail.com