Python2018

compscicenter.ru

aleksey.kladov@gmail.com

Лекция 8

Итераторы

Протокол

```
for x in xs:
    body
```

Протокол

```
for x in xs:
    body

it = xs.__iter__()
while True:
    try:
        x = it.__next__()
    except StopIteration:
        break

    body
```

Iterator

```
class Iterator:
    def __next__(self):
        if self.has_more_elements():
            return self.next_element()
        raise StopIteration


it = Iterator()
elem = next(it, default)
```

Iteratable

```
class Iterable:
    def __iter__(self):
        return Iterator()


x = Iterable()

it = iter(x)  # calls x.__iter__()
```

Iterator is Iteratable

```
class Iterator:
    def __next__(self):
        ...

    def __iter__(self):
        return self
```

```python
class range:
    def __init__(self, start, stop):
        self.start = start
        self.stop = stop

    def __iter__(self):
        return RangeIterator(self.start, self.stop)


class RangeIterator:
    def __init__(self, start, stop):
        self.start = start
        self.stop = stop

    def __iter__(self):
        return self

    def __next__(self):
        if self.start < self.stop:
            res = self.start
            self.start += 1
            return res
        raise StopIteration
```

in Iterator

```
class range:
    ...


class RangeIterator:
    ...


r = range(0, 100)
assert not hasattr(r, "__contains__")
assert 42 in r  # O(N)
```

Итерация два раза

```
>>> r = range(0, 100)
>>> sum(r)
4950
>>> sum(r)
4950
>>> it = iter(r)
>>> sum(it)
4950
>>> sum(it)
0
```

iter

```
def range(start, stop):
    def step():
        nonlocal start
        res = start
        start += 1
        return res

    # iter(step_fn, sentinel)
    return iter(step, stop)
```

Генератор

```python
def range(start, stop):
    while start < stop:
        yield start
        start += 1
```

Генератор

```
>>> def g():
...     print("started")
...     x = 42
...     yield x
...     print("yielded once")
...     x += 1
...     yield x
...     print("yielded twice, done")
...
>>> it = g()
>>> for x in it:
...     print(x)
...
started
42
yielded once
43
yielded twice, done
```

Генератор

```
>>> def g():
...     print("started")
...     x = 42
...     yield x
...     print("yielded once")
...     x += 1
...     yield x
...     print("yielded twice, done")
...
>>> type(g)
<class 'function'>
>>> type(g())
<class 'generator'>
```

Применения

```
def unique(xs):
    seen = set()
    for item in xs:
        if item in seen:
            continue
        seen.add(item)
        yield item


xs = [1, 1, 2, 3]
assert list(unique(xs)) == [1, 2, 3]
```

Применения

```python
def map(func, xs, *rest):
    for args in zip(xs, *rest):
        yield func(*args)

xs = [1, 2, 3, 4]
assert list(map(lambda x: x * x, xs)) \
    == [1, 4, 9, 16]
```

Применения

```
def chain(*xss):
    for xs in xss:
        for item in xs:
            yield item
```

Применения

```python
def chain(*xss):
    for xs in xss:
        yield from xs

xs = [1, 2, 3]
ys = [92]

assert list(chain(xs, ys)) == [1, 2, 3, 92]
```

Применения

```python
def count(start=0):
    while True:  # бесконечный генератор!
        yield start
        start += 1
```

```python
class BinaryTree:
    def __init__(self, value, left=None, right=None):
        self.value = value
        self.left = left
        self.right = right

    def __iter__(self):
        return self.pre_order

    @property
    def pre_order(self):
        yield self
        if self.left:
            yield from self.left
        if self.right:
            yield from self.right

    @property
    def post_order(self):
        ...
```

```python
class BinaryTree:
    def __init__(self, value, left=None, right=None):
        self.value = value
        self.left = left
        self.right = right

    def for_each(self, f):
        f(self)
        if self.left:
            self.left.for_each(f)
        if self.right:
            self.right.for_each(f)
```

```
NOT_VISITED = 0
VISITED_LEFT = 1
VISITED_BOTH = 2

class TreeIter:
    def __iter__(self, root):
        self.stack = [(root, NOT_VISITED)]

    def __next__(self):
        while self.stack:
            node, state = self.stack.pop()
            if state == NOT_VISITED:
                self.stack.append((node, VISITED_LEFT))
                self.stack.append((node.left, NOT_VISITED))
                return node

            if state == VISITED_LEFT
                ...

            if state == VISITED_BOTH:
                ...

        raise StopIteration
```

Выражения генераторы

```
>>> map(lambda x: x * x, xs)
<map object at 0x7ff7437c3ac8>
>>> (x * x for x in xs)
<generator object <genexpr> at 0x7ff7437bbeb8>

# map(lambda и filter(lambda всегда длиннее!

>>> sum(x**2 for x in range(10))  # нет  ()
285

>>> map(lambda s: len(s), ["", "a", "foo"])  # :(
<map object at 0x7effed29dda0>
>>> map(len, ["", "a", "foo"])  # ok!
<map object at 0x7effed29de48>
```

itertools ❤️

```python
from itertools import islice

xs = range(10)

assert list(islice(xs, 2, 8, 3)) == [2, 5]

# islice работает с любым итератором, лениво
```

itertools ❤️

```
from itertools import islice

xs = range(10)

assert list(islice(xs, 2, 8, 3)) == [2, 5]

# islice работает с любым итератором, лениво
# магии нет, просто пропускает элементы

>>> sum(range(0, 10**9, 10**8))
4500000000  # быстро
>>> sum(islice(range(10**9), 0, None, 10**8))
4500000000  # долго
```

itertools ❤️

```python
from itertools import islice

def take(n, xs):
    return list(islice(xs, 0, n))
```

itertools ❤️

```
from itertools import count, cycle, repeat, islice


def take(n, xs):
    return list(islice(xs, 0, n))


assert take(3, count(start=1, step=2)) == [1, 3, 5]
assert take(3, cycle(["любит", "не любит"])) \
    == ["любит", "не любит", "любит"]

assert take(3, repeat(92)) \
    == list(repeat(92, times=3))


#  вторая форма iter -- в builtins
```

itertools ❤️

```
from itertools import dropwhile, takewhile

assert list(dropwhile(lambda x: x < 5, range(10))) \
    == [5, 6, 7, 8, 9]
```

itertools ❤️

```
from itertools import chain

assert list(chain(range(2), "abc")) \
       == [0, 1, "a", "b", "c"]

xs = [range(0, i) for i in range(5)]
assert list(chain(*xs)) \
    == chain.from_iterable(xs) \
    == [0, 0, 1, 0, 1, 2, 0, 1, 2, 3]
```

itertools ❤️

```
from itertools import chain, count, islice

xs = (range(0, i) for i in count())
assert sum(
    chain.from_iterable(islice(xs, 1000))
) == 1165167000
```

itertools ❤️

```
it = iter(range(3))
a = b = it
assert list(a) == [0, 1, 2]
assert list(b) == []
```

itertools ❤️

```
from itertools import tee

it = iter(range(3))
a, b, c = tee(it, 3)
assert list(a) == list(b) == list(c) \
       == [0, 1, 2]

# Магии нет, в худшем случае -- O(N) памяти!
```

itertools ❤️

```python
from itertools import product, combinations, permutations
from itertools import combinations_with_replacement

assert list(product("AB", repeat=2)) \
       == [("A", "A"), ("A", "B"), ("B", "A"), ("B", "B")]

for x in range(3):
    for y in range(5):
        pass

for x, y in combinations(range(5), 2):
    pass
```

itertools ❤️

```
from itertools import combinations

def build_graph(words, mismatch_percent):
    for (i, u), (j, v) in combinations(enumerate(words), 2):
        if len(u) != len(v):
            continue
        ...
```

itertools ❤️

```
from itertools import groupby


def sorted_runs(xs):
    indices = range(len(xs) - 1)

    def is_increasing(idx):
        return xs[idx] < xs[idx + 1]

    return groupby(indices, is_increasing)


xs = [1, 2, 3, 5, 2, 0, 3, 1]
for is_increasing, group in sorted_runs(xs):
    print(
        "<" if is_increasing else ">",
        sum(1 for _ in group),  # Есть магия, O(1) памяти!
    )
```

Генераторы и IO

```
from contextlib import ExitStack
import heapq


def merge_sorted_files(inputs, result):
    with open(result, 'w') as result, \
            ExitStack() as stack:
        files = [stack.enter_context(open(f))
                 for f in inputs]
        for line in heapq.merge(*files):
            result.write(line)


merge_sorted_files(["10GB.txt", "20GB.txt"], "merged.txt")
```

Генераторы -- паттерн

```
import os

class cd:
    def __init__(self, path):
        self.path = path

    def __enter__(self):
        self.saved_cwd = os.getcwd()
        os.chdir(self.path)

    def __exit__(self, *exc_info):
        os.chdir(self.saved_cwd)
```

Генераторы -- паттерн

```
import os
from contextlib import contextmanager

@contextmanager
def cd(path):                       # __init__
    old_path = os.getcwd()          # __enter__
    os.chdir(path)
    try:
        yield                       # ---------
    finally:
        os.chdir(old_path)          # __exit__
```

Генераторы -- паттерн

```python
import tempfile
import shutil

@contextmanager
def tempdir():                          # __init__
    outdir = tempfile.mkdtemp()         # __enter__
    try:
        yield outdir                    # ---------
    finally:
        shutil.rmtree(outdir)           # __exit__
```

Генераторы -- паттерн

```
import tempfile
import shutil

@contextmanager
def tempdir():
    outdir = tempfile.mkdtemp()
    try:  # но как???
        yield outdir
    finally:
        shutil.rmtree(outdir)
```

Корутины

```python
def sum_and_sumsq(it):
    s = s2 = 0
    for item in it:
        s += item
        s2 += item*item
    return s, s2
```

Корутины

```
import itertools


def sum_powers(it, p):
    acc = 0
    for item in it:
        acc += item ** p
    return acc


def sum_and_sumsq(it):
    it1, it2 = itertools.tee(it, 2)
    s = s2 = 0
    return sum_powers(it1, 1), sum_powers(it2, 2) # :(
```

Корутины

```
>>> def printer():
...     while True:
...         item = yield
...         print(item)
...
>>> p = printer()
>>> p.send(None)  # next(p)
>>> p.send(92)
92
```

Корутины

```
>>> def running_sum():
...     acc = 0
...     while True:
...         acc += yield acc
...
>>> s = running_sum()
>>> s.send(None)
0
>>> s.send(1)
1
>>> s.send(1)
2
>>> s.send(1)
3
```

Корутины

```python
import functools


def coroutine(func):
    @functools.wraps(func)
    def inner(*args, **kwargs):
        gen = func(*args, **kwargs)
        next(gen)
        return gen

    return inner
```

Корутины

```
import functools


def coroutine(func):
    @functools.wraps(func)
    def inner(*args, **kwargs):
        gen = func(*args, **kwargs)
        next(gen)
        return gen

    return inner
```

Корутины

```
@coroutine
def logger():
    with open("log.txt") as f:
        while True:
            item = yield
            print(item, file=f)


l = logger()
l.send(92)
del l  # oops?
```

Корутины

```
@coroutine
def printer():
    try:
        while True:
            item = yield  # raises GeneratorExit
            print(item)
    finally:
        print("Cleaning up")


p = printer()
p.send(92)  # 92
p.close()   # Cleaning up
```

Корутины

```
@coroutine
def sum_powers_coro(p):
    acc = 0
    while True:
        item = yield
        acc += item ** p
```

Корутины

```
STOP = object()


@coroutine
def sum_powers_coro(p):
    acc = 0
    while True:
        item = yield
        if item is STOP:
            return acc
        acc += item ** p
```

Корутины

```
STOP = object()


def result(coro):
    try:
        coro.send(STOP)
    except StopIteration as e:
        return e.value


@coroutine
def sum_powers_coro(p):
    acc = 0
    while True:
        item = yield
        if item is STOP:
            return acc
        acc += item ** p
```

Корутины

```
def sum_and_sumsq(it):
    s = sum_powers_coro(p=1)
    s2 = sum_powers_coro(p=2)
    for item in it:
        s.send(item)
        s2.send(item)
    return result(s), result(s2)


assert sum_and_sumsq(iter(range(10))) == (45, 285)
```

Корутины

```python
import tempfile
import shutil

@contextmanager
def tempdir():
    outdir = tempfile.mkdtemp()
    try:  # но как???
        yield outdir
    finally:
        shutil.rmtree(outdir)
```

```
class Manager(AbstractContextManager):
    def __init__(self, co):
        self._co = co

    def __enter__(self):
        return next(self._co)

    def __exit__(self, exc_type, exc_val, exc_tb):
        if exc_val is not None:
            try:
                self._co.throw(exc_val)
            except StopIteration:
                pass
        return True


def contextmanager(f):
    @functools.wraps(f)
    def inner(*args, **kwargs):
        return Manager(f(*args, **kwargs))

    return inner
```

Корутины

- .send, .throw, .close посылают значение или исключение в генератор
- __next__ это .send(None)
- генераторы это больше, чем итераторы

Почитать в транспорте

https://docs.python.org/3.7/library/itertools.html