Python2018

compscicenter.ru

aleksey.kladov@gmail.com

Лекция 5

Коллекции и collections

Списки, напоминание

- внутри -- расширяющийся массив
- O(1) индексация
- Добавление/Удаление с конца -- O(1)
- Копирование -- O(N)
- Не копируйте списки!

Стэк

```
>>> xs = [1, 2, 3]
>>> xs.append(4)
>>> xs.pop()
4
```

Связный список

- всегда хуже расширяющегося массива
- удаление из середины
  - O(N) на поиск середины
  - O(1) на удаление
  - = O(N), и медленнее, чем массив
- нет в Python :)
- https://www.youtube.com/watch?v=YQs6IC-vgmo

reverse[d]

```
>>> xs = [1, 2, 3]
>>> xs.reverse()   # verb
>>> xs
[3, 2, 1]
>>> reversed(xs)   # adjective
<list_reverseiterator object at 0x7fd575c05908>
>>> list(reversed(xs))  # копия списка!
[1, 2, 3]
>>> xs
[3, 2, 1]
```

sort[ed]

```
>>> xs = [92, 42, 62]
>>> sorted(xs)
[42, 62, 92]
>>> xs
[92, 42, 62]
>>> xs.sort()
>>> xs
[42, 62, 92]
>>> xs = ["Greenaway", "Adams", "Cortázar"]
>>> sorted(xs, key=len, reverse=True)
['Greenaway', 'Cortázar', 'Adams']
```

key vs cmp

- key: (obj) -> key value
- cmp: (obj_a, obj_b) -> LT | EQ | GT
- key -- почти всегда более естественное API
- functools.cmp_to_key

Составной ключ

```
>>> (1, 2) < (3, 0)
True
>>> (1, 2) < (1, 2,  3)
True
>>> sorted(
...     people,
...     key=lambda p: (p.first_name, -len(p.last_name))
... )
[...]
>>> people.sort(key: lambda p: p.last_name, reversed=True)
>>> people.sort(key: lambda p: p.first_name)
```

Очередь

Это не очередь

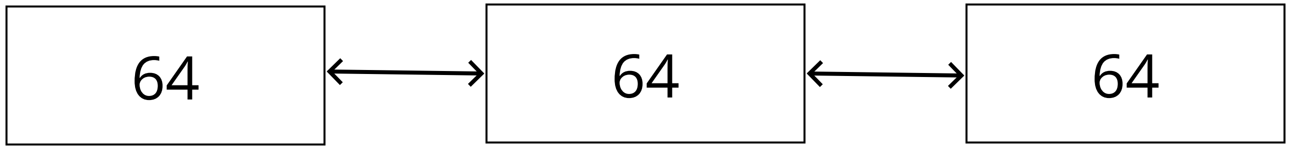
```python
from queue import Queue
```

The queue module implements multi-producer, multi-consumer queues. It is especially useful in **threaded programming** when information must be exchanged safely between multiple threads.

deque

```
>>> from collections import deque
>>> q = deque([3, 4, 5])
>>> q
deque([3, 4, 5])
>>> q.appendleft(2)
>>> q.pop()
5
>>> q
deque([2, 3, 4])
>>> q.rotate(1)
>>> q
deque([4, 2, 3])
>>> q.maxlen
>>>
```

deque



- linked list of chunks
- O(N) индексация

Очередь с приоритетом

```
>>> import heapq
>>> import random
>>> import string
>>> xs = [(random.randrange(10), c)
...       for c in string.ascii_uppercase[:5]]
>>> xs
[(3, 'A'), (1, 'B'), (5, 'C'), (8, 'D'), (0, 'E')]
>>> heapq.heapify(xs)
>>> xs
[(0, 'E'), (1, 'B'), (5, 'C'), (8, 'D'), (3, 'A')]
>>> heapq.heappop(xs)
(0, 'E')
>>> xs
[(1, 'B'), (3, 'A'), (5, 'C'), (8, 'D')]
>>> heapq.heappush(xs, (2, 'X'))
>>> xs
[(1, 'B'), (2, 'X'), (5, 'C'), (8, 'D'), (3, 'A')]
```

Очередь с приоритетом

```
def merge(*iterables, key=None, reverse=False):
    '''Merge multiple sorted inputs into a single sorted output.
       ...
    '''
```

Кортежи

```
>>> person = ("George", "Carlin", "May", 12, 1937)
>>> last_name = person[1]
>>> birthday = person[2:]
>>> last_name
'Carlin'
>>> birthday
('May', 12, 1937)
```

Кортежи

```
>>> person = ("George", "Carlin", "May", 12, 1937)
>>> LAST_NAME = 1
>>> BIRTHDAY = slice(2, None)  # [2:]
>>> person[LAST_NAME]
'Carlin'
>>> person[BIRTHDAY]
('May', 12, 1937)
```

Кортежи

```
>>> from collections import namedtuple
>>> Person = namedtuple(
...     "Person",
...     ["first_name", "last_name", "age"],
... )
>>> p = Person("Terrence", "Gilliam", 77)
>>> (p.first_name, p.last_name, p.age)
('Terrence', 'Gilliam', 77)
>>> p._replace(first_name="Terry")
Person(first_name='Terry', last_name='Gilliam', age=77)
>>> p[2]
77
```

And Now For Something Completely Different

```python
from typing import List


def fib(n: int) -> List[int]:
    fib1: int = 1
    fib2: int = 1
    res = []
    for _ in range(n):
        res.append(fib1)
        fib1, fib2 = fib2, fib1 + fib2
    return res
```

And Now For Something Completely Different

```python
from typing import NamedTuple


class Person(NamedTuple):
    first_name: str
    last_name: str
    age: int = 42


p = Person("Terrence", "Gilliam", 77)
```

NamedTuple

- не изменяемый
- hash/eq
- ord
- __repr__

One way to do it

```python
from dataclasses import dataclass  # >= 3.7


@dataclass(frozen=True)
class Person:
    first_name: str
    last_name: str
    age: int = 42
```

Словари

```
graph = {}


def add_edge(u, v):
    if u not in graph:
        graph[u] = []
    graph[u].append(v)


def neighbours(u):
    return graph[u] if u in graph else []
```

Словари

```
graph = {}


def add_edge(u, v):
    graph.setdefault(u, []).append(v)


def neighbours(u):
    return graph.get(u, [])
```

Словари

```
>>> from collections import defaultdict
>>> graph = defaultdict(list)
>>> graph[1].append(2)
>>> graph[1].append(3)
>>> graph[1]
[2, 3]
>>> graph[2]
[]
```

Словари

```
from collections import defaultdict


def count_words(text):
    res = defaultdict(lambda: 0)  # int
    for word in text.split():
        res[word] += 1
    return res


words = sorted(
    count_words(open(__file__).read()).items(),
    key=lambda it: it[1],
    reverse=True,
)
for word, count in words[:3]:
    print(f"{word:<5}: {count}")
```

Словари

```
from collections import Counter


def count_words(text):
    return Counter(text.split())


words = count_words(open(__file__).read())
for word, count in words.most_common(3):
    print(f"{word:<5}: {count}")
```

Словари

```
>>> from collections import Counter
>>> c = Counter(a=3, b=1)
>>> d = Counter(a=1, b=2)
>>> c + d
Counter({'a': 4, 'b': 3})
>>> c - d
Counter({'a': 2})
>>> c & d  # пересечение:  min(c[x], d[x])
Counter({'a': 1, 'b': 1})
>>> c | d  # объединение:  max(c[x], d[x])
Counter({'a': 3, 'b': 2})
>>>
```

Словари

```
>>> c = Counter()
>>> c[92] = -10
>>> c[92] += 1
>>> c
Counter({92: -9})
>>> c.update([92])
>>> c
Counter({92: -8})
>>> +c
Counter()
>>> c
Counter({92: -8})
```

Словари

```
>>> from collections import ChainMap
>>> locals = {}
>>> globals = {'foo': 92}
>>> builtins = {'baz': 93}
>>> scope = ChainMap(locals, globals, builtins)
>>> scope['foo']
92
>>> scope['foo'] = 1
>>> scope['foo']
1
>>> globals
{'foo': 92}
>>> locals
{'foo': 1}
>>> scope.maps
[{'foo': 1}, {'foo': 92}, {'baz': 93}]
```

Словари

```
>>> locals = {}
>>> globals = {'foo': 92}
>>> builtins = {'baz': 93}
>>> scope = {**locals, **globals, **builtins}
```

Словари

```
>>> locals = {}
>>> globals = {'foo': 92}
>>> builtins = {'baz': 93}
>>> scope = {**builtins, **globals, **locals}
```

Словари

```
>>> from collections import OrderedDict
>>> d = OrderedDict()
>>> d['foo'] = 1
>>> d['bar'] = 2
>>> list(d)
['foo', 'bar']  # порядок гарантирован
```

Словари

Начиная с ~3.6, OrderedDict ~= dict

Словари

```
>>> from collections import OrderedDict
>>> a = OrderedDict()
>>> a["foo"] = 1
>>> a["bar"] = 2
>>> b = OrderedDict()
>>> b["bar"] = 2
>>> b["foo"] = 1
>>> a
OrderedDict([('foo', 1), ('bar', 2)])
>>> b
OrderedDict([('bar', 2), ('foo', 1)])
>>> a == b
False
>>> c = {"foo": 1, "bar": 2}
>>> a == c
True
>>> c == b
True
>>> "Practicality beats purity"
```

Почитать в транспорте

https://docs.python.org/3/library/collections.html