Python Data Structures

Zbigniew Jędrzejewski-Szmek

George Mason University

Python Summer School, Zürich, September 02, 2013

Version `Zürich-83-gf79f32d`

Outline

1. Introduction (using lists)
    - How does Python manage memory?
    - Comprehensions
2. Dictionaries, and sets
    - Dictionaries
    - How to create a dictionary?
    - Specialized dictionary subclasses
    - Sets
3. Iterators
    - Iterator classes
    - Generator expressions
    - Generator functions
4. The end

Introduction (and lists)

Why?

You can write `sort()`, but it'll take one day.
DRY. Use proper data structures.

The venerable `list`

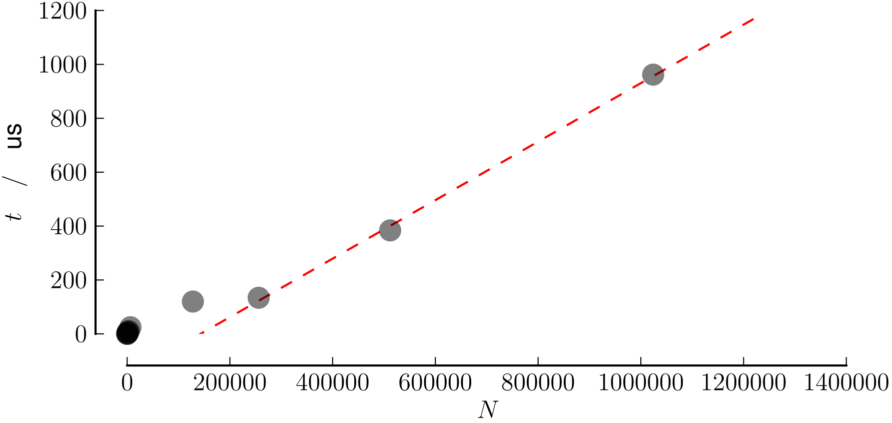
A sequence of arbitrary values

Quiz:
Is `list.pop(0)` faster than `list.pop()`?

Test, test, test

```
N = 1000
L0 = range(N)
L = L0[:]
for i in xrange(N):
    L.pop(0)
```

How does time required depend on problem size?



Reminder: computational complexity

$$f(x) = O(g(x))$$

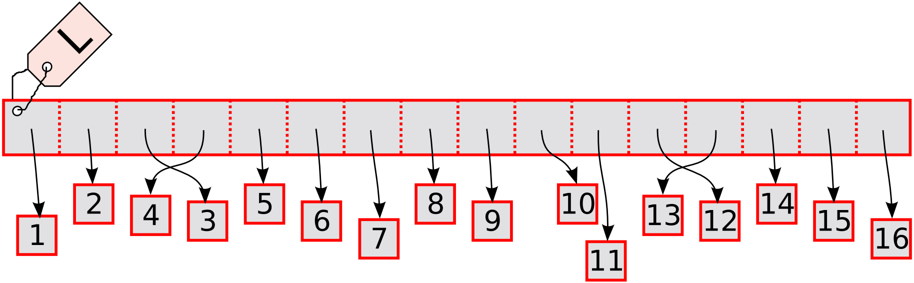
There's a constant C, such that
$f(x) \leqslant C \cdot g(x)$ for big enough x.

For “computational complexity” x is N

What does this mean?

Computational complexity depends on the data structure and operation — good to understand what you're doing

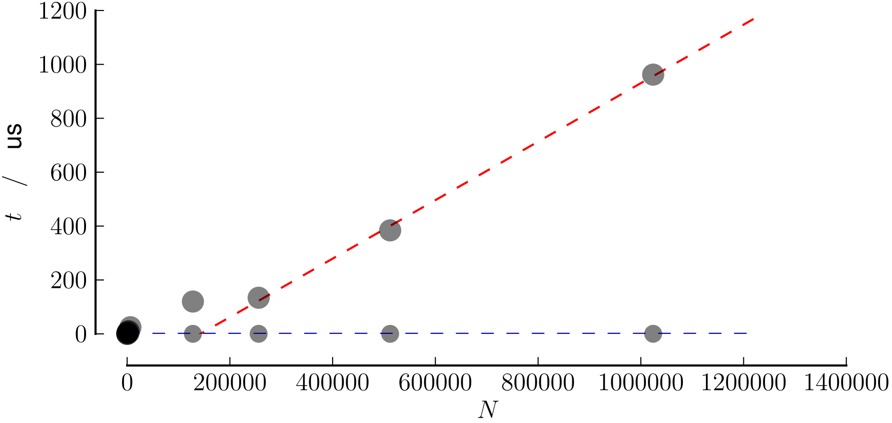
How is `list` implemented?



`list.pop(0)`

`list.pop(-1)`

pop(0) vs pop(-1)


1200
1000
800
600
400
200
0
t / us
0
200000
400000
600000
800000
1000000
1200000
1400000
N

Solution

for the `.pop(0)` problem

`collections.deque` pops from both ends cheaply:

- .pop()
- .popleft()

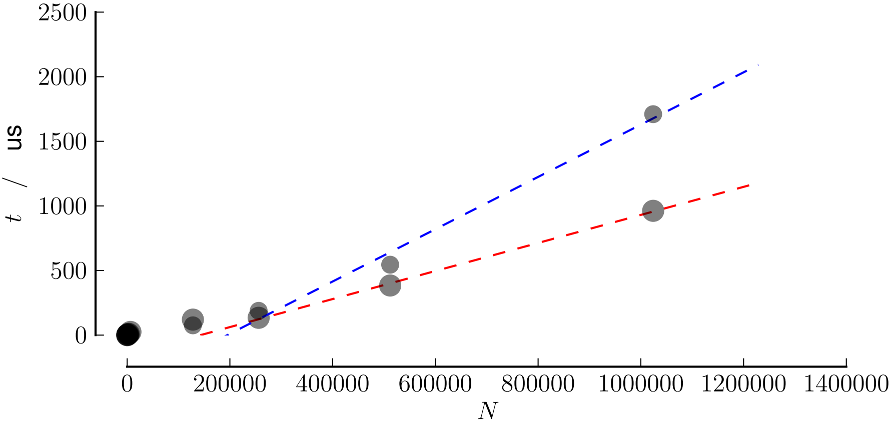
. . . but does not allow other removals in the middle

The complexity constant

$$f(x) \leqslant \mathbf{C} \cdot g(x)$$

Complexity constant example

CPython vs. Jython



Testing is not enough here

Time complexity issues
do not show up during development,
because $O(N^2)$ is small for small N.

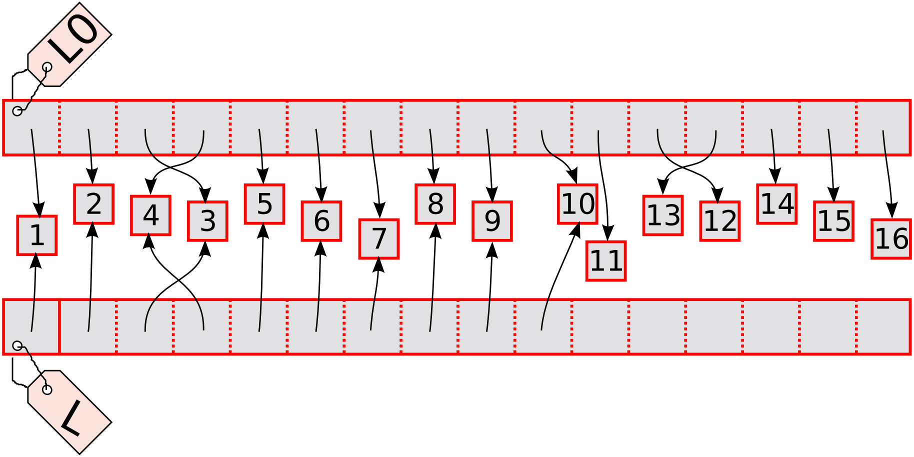
Removing an element from a list

```
N = 1000
L0 = range(N)
L = L0[:]
...
```

Why?

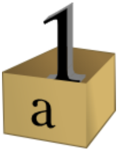
Objects are kept around if they are referenced

`ints` are referenced from both `L` and `L0`

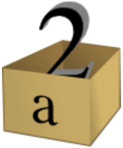


Other languages have "variables"

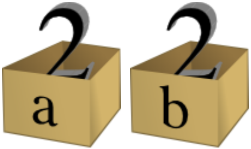
```
int a = 1;
```



```
a = 2;
```

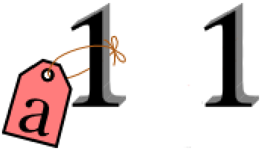


```
int b = a;
```

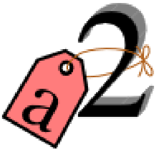


Python has "names"

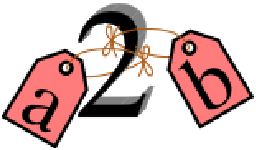
```
a = 1
```



```
a = 2
```



```
b = a
```



List comprehensions

```
[day for day in week]

[day for day in week if day[0] != ’M’ ]

[(day, hour) for day in week
             for hour in [’12:00’, ’14:00’, ’16:00’] ]
```

Dictionaries and sets

Creating dictionaries

dict == a mapping of keys to arbitrary values

```
D = {
  ’key1’: "value1",
  2: [’a’, ’list’, ’here’],
  None: None,
}
```

From an iterator

```
text = """\
lundi     Monday
mardi     Tuesday
mercredi  Wednesday
jeudi     Thursday
vendredi  Friday
samedi    Saturday
dimanche  Sunday
"""

source = (line.split() for line in text.splitlines())

weekdays = dict(source)
```

From a comprehension

```
{x:x**2 for x in xrange(10)}
```

How much does it cost to enter an item in the dictionary?

```
>>> D = {}
>>> L = range(10000)
>>> for i in L:
...     D[i] = i
```

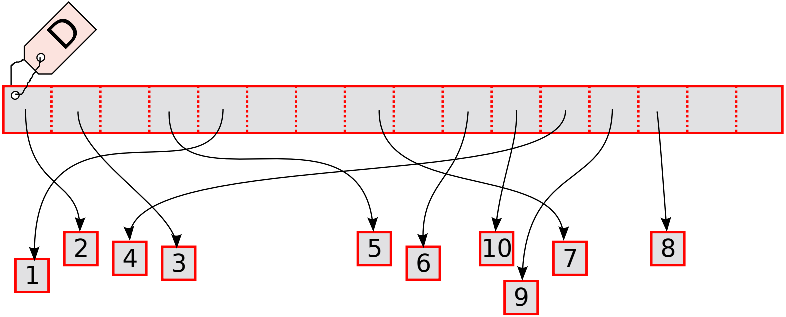
$$O^*(1)$$

How much does it cost to retrieve an item from the dictionary?

```
>>> for i in L:
...     assert D[i] == i
```

$$O(1)$$

How are dictionaries organized?



Weekdays in French and in English

The order of keys is random!

A list of the weekdays?

```
>>> weekdays.keys()
['mardi', 'samedi', 'vendredi', 'jeudi',
 'lundi', 'dimanche', 'mercredi']
```

Demo

hash_demo.py

```python
# 1. Initialize a dict with the same keys.
# 2. Display keys.

from __future__ import print_function

d = dict(monday=1, tuesday=2, wednesday=3,
         thursday=4, friday=5)
print(d.keys())
print('monday ->', hash('monday'))
```

collections.OrderedDict

```
# source = (('lundi', 'Monday'),
#           ('mardi', 'Tuesday'),
#           ...

>>> weekdays = collections.OrderedDict(source)

>>> weekdays.keys()
['lundi', 'mardi', 'mercredi', 'jeudi',
 'vendredi', 'samedi', 'dimanche']
```

Classifying objects

Sorting objects into groups
grades.log:

```
20120101 John 5
20120102 John 4
20120107 Mary 3
20120109 Jane 2
...
```

Sorting objects into groups

version 0

```
grades = {}
for line in open('grades.log'):
    date, person, grade = line.split()
    grades[person].append(grade)
```

Sorting objects into groups

version 0.5

```
grades = {}
for line in open('grades.log'):
    date, person, grade = line.split()
    if person not in grades:
        grades[person] = []
    grades[person].apend(grade)
```

Sorting objects into groups

version 1.0

```
grades = collections.defaultdict(list)
for line in open('grades.log'):
    date, person, grade = line.split()
    grades[person].append(grade)
```

Counting objects in groups

Let’s make a histogram

```
grades = collections.Counter()
for line in open(’grades.log’):
    date, person, grade = line.split()
    grades[grade] += 1
```

```
>>> grades
Counter({’4’: 2, ’3’: 1, ’2’: 1, ’5’: 1})
```

```
>>> print(’\n’.join(x + ’ ’ + ’x’*y
...         for (x, y) in sorted(grades.items())))
```

```
1 x
2 xxx
3 x
4 xxx
5 xx
```

Set

A non-ordered collection of unique elements

```
visitors = set()

for connection in connection_log():
    ip = connection.ip
    if ip not in visitors:
        print('new visitor', ip)
        visitors.add(ip)
```

Using sets

Poetry:
find words used by the poet, sorted by **length**

```
>>> shak = ’’’\
... She walks in beauty, like the night
... Of cloudless climes and starry skies;
... And all that’s best of dark and bright
... Meet ...
... ’’’
```

```
>>> words = set(shak.lower().translate(None, ’,;’).split())
>>> words
{’she’, ’like’, ’cloudless’, ...}
```

```
>>> sorted(words, key=len, reverse=True)
[’cloudless’, ’starry’, ’beauty’, ...]
```

Iterators

Collections and their iterators

- $\boxed{\text{sequence}}.\_\_\text{iter}\_\_() \longrightarrow \boxed{\text{iterator}}$
- $\boxed{\text{iterator}}.\_\_\text{iter}\_\_() \longrightarrow \text{self}$
- $\boxed{\text{iterator}}.\text{next}() \longrightarrow \boxed{\text{item}}$
- $\boxed{\text{iterator}}.\text{next}() \longrightarrow \boxed{\text{item}}$
- $\boxed{\text{iterator}}.\text{next}() \longrightarrow \boxed{\text{item}}$

Iterators can be non-destructive or destructive

- `list`
- `file`

How can we create an iterator?

1. write a class with `.__iter__` and `.next`
2. use a generator expression
3. write a generator function

1. Iterator class

```
import random

class randomaccess(object):
    def __init__(self, alist):
        self.indices = range(len(alist))
        random.shuffle(self.indices)
        self.alist = alist

    def next(self):
        return self.alist[self.indices.pop()]

    def __iter__(self):
        return self
```

2. Generator expressions

```
# chomped lines
(line.rstrip() for line in open(some_file))
```

```
# remove comments
(line for line in open(some_file)
      if not line.startswith('#'))
```

```
>>> type([i for i in 'abc'])
<type 'list'>
>>> type( i for i in 'abc' )
<type 'generator'>
```

3. Generator functions

```
>>> def countdown(n):
...   print '--start'
...   for i in range(n, 0, -1):
...     yield i
...   print '--done'
>>> for i in countdown(3):
...   print i
--start
3
2
1
--done
```

```
>>> g = gen(2)
>>> g
<generator object ...>
>>> g.next()
--start
2
>>> g.next()
1
>>> g.next()
--stop
Traceback (most recent cal
  ...
StopIteration
```

Generator objects

__iter__ and next

```
>>> g.__iter__()
<generator object gen at 0x7f24a5e1c690>
>>> g
<generator object gen at 0x7f24a5e1c690>

>>> g.next()
--start
1
```

That's all!