Контейнеры

- Массив(список) list
- Константный массив(кортеж) tuple
- Словарь dict
- Множество set
- frozenset, buffer, ...

```
1        [1, 2, 3]
2        (1, 2, 3)
3        {1:"1", 2:"2", 3:"3"}
4        {1, 2, 3}
```

list – Список (Массив)

- Упорядоченное множество элементов, доступ по номеру
- var = [1, 2, 3]
- Индексация arr[x]
- Срезы arr[frm:to:step]
  [arr[frm], arr[frm + step], ...,]
- Отрицательный индекс - отсчет от конца к началу. x[-1]
- Отсутвие индекса - frm $\rightarrow$ 0, to $\rightarrow$ len(x), step $\rightarrow$ 1

list – Список (Массив)

$x = [0^{0}_{-6}, 1^{1}_{-5}, 2^{2}_{-4}, 3^{3}_{-3}, 4^{4}_{-2}, 5^{5}_{-1}]$

x[2]	== 2	[0, 1, 2, 3, 4, 5]
x[-2]	== 4	[0, 1, 2, 3, 4, 5]
x[2:]	== [2, 3, 4, 5]	[0, 1, 2, 3, 4, 5]
x[-2:]	== [4, 5]	[0, 1, 2, 3, 4, 5]
x[1:-1]	== [1, 2, 3, 4]	[0, 1, 2, 3, 4, 5]
x[1:-1:2]	== [1, 3]	[0, 1, 2, 3, 4, 5]
x[1:-1:-2]	== [4, 2]	[0, 1, 2, 3, 4, 5]
x[::-1]	== [5, 4, 3, 2, 1, 0]	

list – Операции нам элементам и срезам

```
x = [3, 4, 5, 6]
range(x) == [0, ..., x - 1]
range(x, y, z) ~= range(INF)[x:y:z]

x[::2] = [2, 2]
x == [2, 4, 2, 6]
x[::2] = 2 # TypeError
x[:2] = [7] # x == [7, 2, 6]

del x[1] # x == [7, 6]

x = [1, None, True, ["123", 2.4]]
[1, 2, 3] + ["a", "b"] # [1, 2, 3, "a", "b"]
```

Методы списка

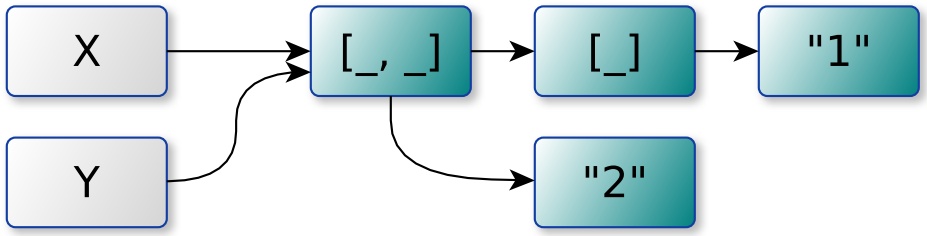
```
# arr.append(val)
[1, 2].append(3) => [1, 2, 3]

# arr.extend(arr2)
[1, 2].extend([2, 3]) => [1, 2, 2, 3]

# arr.pop()
x = [1, 2]
x.pop() == 2
x == [1]

# arr.insert(pos, val)
[1, 2].insert(0, "abc") => ["abc", 1, 2]

[1, 2].index(2) => 1
[1, 2].reverse() => [2, 1]
[1, 2, 4, 1, 2, 4, 1, 1].count(1) == 4
x = [1, 3, 2]
x.remove(1) # x == [3, 2]
x.sort() # x == [2, 3]
```

Изменяемые типы (ссылочные)

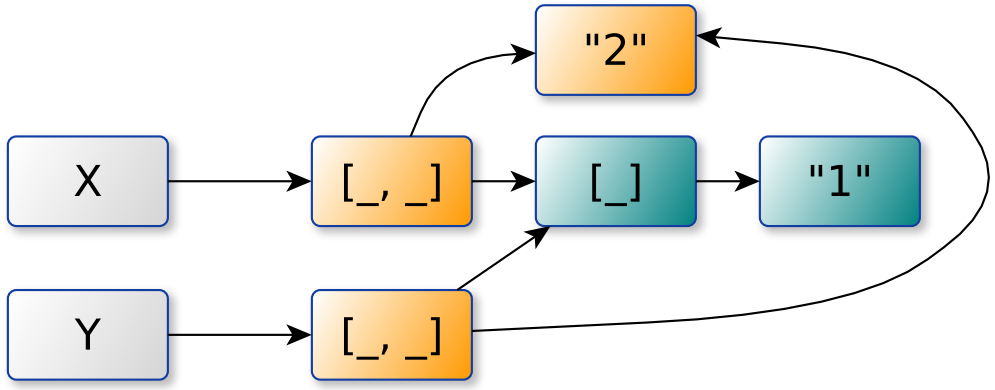
```
1    x = [1, 2, 3]
2    y = x
3    y.append(1)
4    print x => [1, 2, 3, 1]
```

Изменяемые типы (ссылочные)

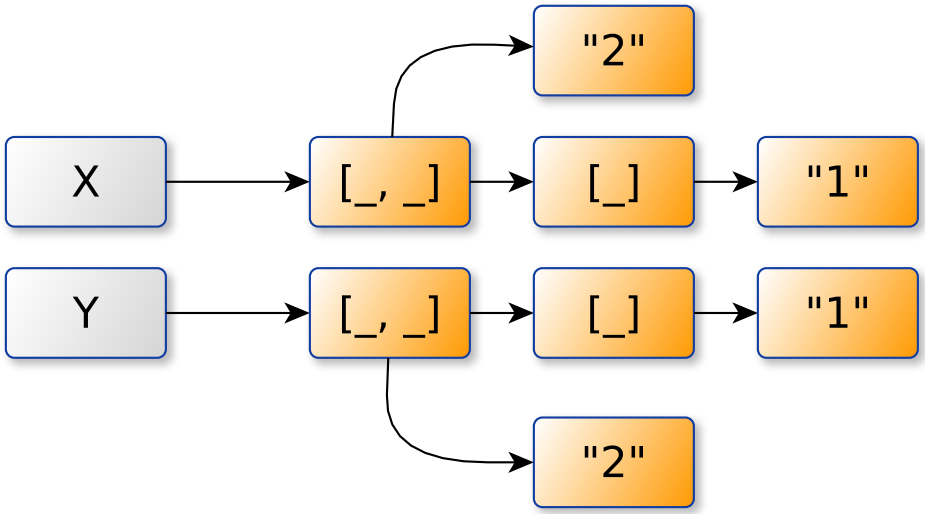
```
1       x = [["1"], "2"]
2       y = x
```




```
1       x = [["1"], "2"]
2       y = x[:]
```



```
1       import copy
2       x = [["1"], "2"]
3       y = copy.deepcopy(x)
```



Сортировка

- sort() сортировка на месте, sorted - возвращает копию
- Не надо сортировать неоднородные контейнеры типы - результат не определен
  По больше части - получится что-то, но иногда - ['x', '\xf0', u'x']. sort ()
  - UnicodeDecodeError

```
1  x = [2, 3, 4, 1 , 6, -2]
2  sorted(x) == [-2, 1, 2, 3, 4, 6]
3  x == [2, 3, 4, 1 , 6, -2]
4  x.sort()
5  x == [-2, 1, 2, 3, 4, 6]
```

tuple – кортеж

- Константный список (но можно изменять элементы, если они не константные)
- Предназначается для небольших коллекций разнородных элементов

```
1       tpl = (1, 2)
2       tpl = 1, 2
3       tpl[1] = 3 # TypeError
4       tpl = (1, [2, 3, 4])
5       tpl[1].append(1) => (1, [2, 3, 4, 1])
6       x, y, z  = (1, 2, 3)
7
8       (1) == 1
9       (1,) == (1,)
```

dict - словарь

- Набор пар (ключ, значение), с быстрым поиском по ключу
- Только константные ключи (tuple - ok)
- Элементы неупорядоченны
- Нет срезов

```
1       x = {1:2, "3":4}
2       x[1] == 2
3       x[2] # KeyError
4       1 in x == True
5       x[17] = True
6       x == {1:2, "3":4, 17:True}
```

dict – Словарь

```
x = {1:2, "3":"4"}
dict(a=1, b=2) == {"a":1, "b":2}
x.items() == [(1, 2), ("3", "4")]
x.values() == [2, "4"]
x.keys() == [1, "3"]
x.copy() == {1:2, "3":"4"}
x.setdefault(key, val) == val
x.get(5) == None
x.get(5, 12) == 12
x.clear() # {}
x.update(y)
dict.fromkeys(keys, val)
# {key[0]:val, key[1]:val, ..} default val is None
```

set - множество

- Множество элементов с быстрым поиском и операциями

```
1       x = {1, 2}
2       y = set([2, "a"]) # y == {2, "a"}
3       x & y == x.intersection(y) == {2}
4       x - y == x.difference(y) == {1}
5       x | y == x.union(y) == {1, 2, "a"}
6       x.issubset(y) # ....
7
8       set(1, 2, 3) # error
9       set("abcd") == set(["a", "b", "c", "d"])
```

Особенности поведения set & dict

- Ключи сравниваются с помощью hash, затем ==
- hash(2.0) == 2, hash(2) == 2, 2.0 == 2
- Для пользовательских объектов hash & == можно перегрузить

```
1  {2.0: "ccc", 2: "dd"} == {2.0: "ccc"}
2  set([2]) | set([2.0]) == set([2])
3  set([2.0]) | set([2]) == set([2.0])
4  set([2]) == set([2.0])
```

Общие операции над контейнерами

- Пребразование к list, tuple, set
- len - количество элементнов
- str.join ", ". join (["1", "2", "3"]) == "1, 2, 3"
- in проверка включения элемента
- Проеобразование контейнера пар к словарю

```
y = [1, 2, 3]
set(y) == {1, 2, 3}
tuple(y) == (1, 2, 3)
set(1, 2, 3) # TypeError
x = [(1, "1"), (2, "2")]
dict(x) == {1:"1", 2:"2"}
```

heapq

Массив, где $x_n >= x_{2n}, x_n >= x_{2n+1}$

```
1       x = [2, 4, 5, 3, 8, 1, 7, 9, 0, 6]
2       heapq.heapify(x)
3       x == [0, 2, 1, 3, 6, 5, 7, 9, 4, 8]
4       tuple(y) == (1, 2, 3)
5       set(1, 2, 3) # TypeError
6       x = [(1, "1"), (2, "2")]
7       dict(x) == {1:"1", 2:"2"}
```

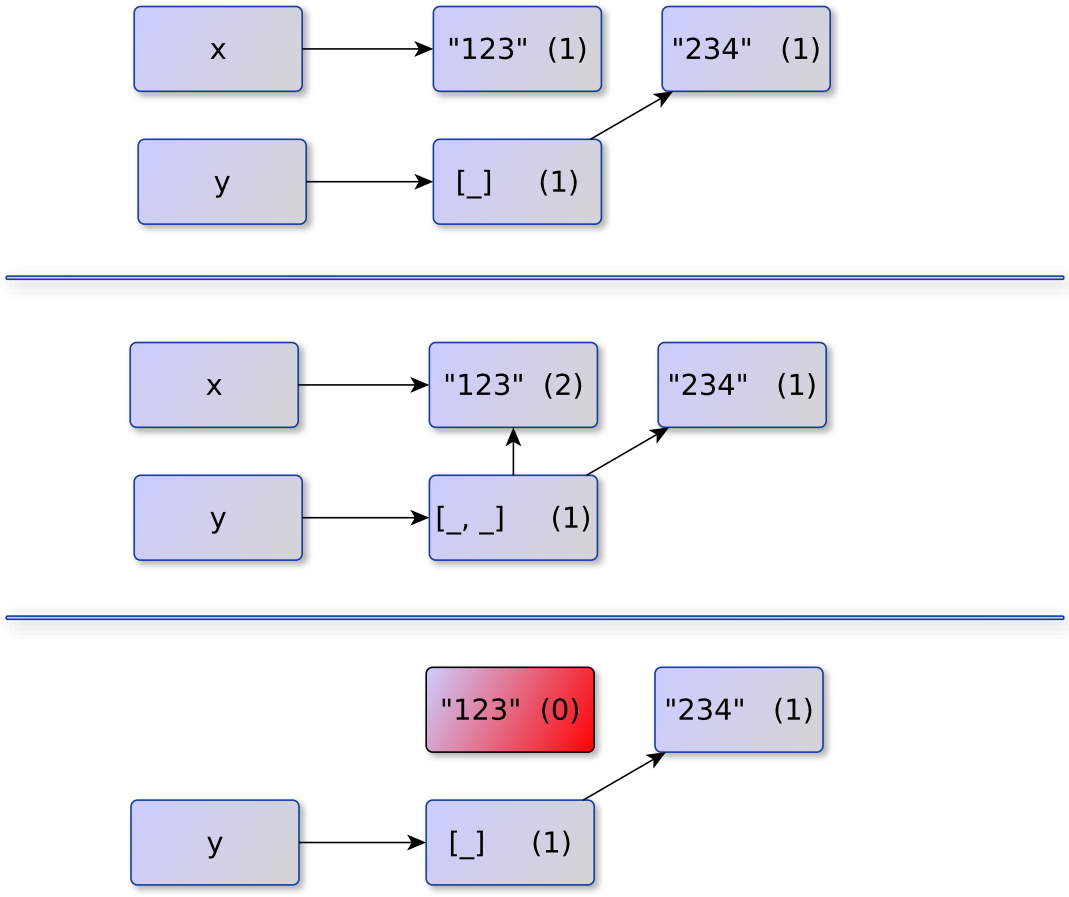
Ассимптотика контейнеров

	list/tuple	dict/set	sorted list	heap
insert/remove begin	$O(n)$	$O(1)^*$	$O(n)$	$O(log(n))$
insert/remove end	$O(1)$	$O(1)^*$	$O(1)$	---
find by key	---	$O(1)^*$	---	---
find by value	$O(n)$	$O(n)$	$O(log(n))$	$O(n)$
find min/max	$O(n)$	$O(n)$	$O(1)$	$O(1)$/ $O(n)$

Сборка мусора

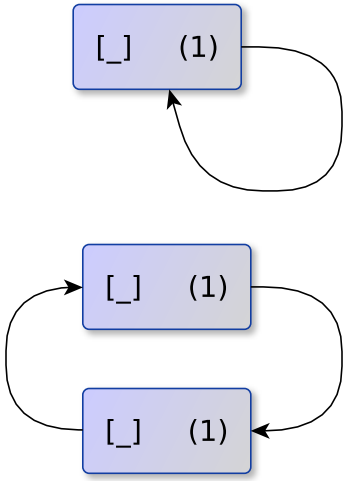
- С каждым объектом связан счетчик ссылок
- Счетчик инкрементируется при появлении каждой новой ссылки на объект: переменной, помещения в контейнер, etc
- И декрементируется при каждом исчезновении ссылки
- Как только счетчик оказывается равен нулю - объект удаляется

Сборка мусора



Циклические ссылки не удаляются автоматически. Такие объекты остаются в памяти до запуска сборщика мусора.

```
1        a = []
2        a.append(a)
3
4        a = []
5        b = []
6        a.append(b)
7        b.append(a)
```



Файлы

- Абстракция для источников или приемников данных
- Тестовые и бинарные
- Поддерживается запись/чтение/позиционирование/os.ioctl
- open(name, mode) - открывает файл
- read([len]) - читает данные. При окончании данных возвращает ""
- write(string) - пишет данные
- seek, tell, readline, readlines

```
1       import os
2
3       fd = open(path, mode)
4       mode in {"r", "w", "r+", "a",
5                "rb", "wb", "rb+", "ab"}
6       fd.read(size) # data str
7       fd.write("data")
8       fd.seek(pos, frm)
9
10      # frm in {os.SEEK_SET, os.SEEK_CUR, os.SEEK_END}
11
12      fd.read() # till the end
13      fd.readline() # untill "\n"
```

Можно итерировать по строкам:

```
1       import os
2
3       fd = open(path)
4       for line in fd:
5               # do something with line
```

```
1       fd = open(path, "r", encoding='utf-8')
2       fd.read(size) # unicode str
3
4       fd = open(path, "rb")
5       fd.read(size) # bytes
```

Модуль io содержит дополнительные классы для работы с воодом-выводом