

## Контейнеры

- Массив(список) `list == [...]`
- Константный массив(кортеж) `tuple == (...)`
- Словарь `dict == {a:b, ...}`
- Множество `set == {a, b, ....}`
- `frozenset, buffer, ....`

## 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. `x[-1]`
- Отсутствие индекса - `frm -> 0`, `to -> -1`, `step -> 1`
- `arr[::-1]` - инверсия элементов
- `arr[:]` - копия

list – Список (Массив)

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

$$x[2] == 2 \quad [0, 1, 2, 3, 4, 5]$$

$$x[-2] == 4 \quad [0, 1, 2, 3, 4, 5]$$

$$x[2:] == [2, 3, 4, 5] \quad [0, 1, 2, 3, 4, 5]$$

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

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

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

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

## list – Операции над элементами и срезами

```
1  x = [3, 4, 5, 6]
2  range(x) == [0, ..., x - 1]
3  range(x, y, z) == range(x)[y:z]
4
5  x[::2] = [2, 2] # x == [2, 4, 2, 6]
6  x[::2] = 2 # error
7  del x[1] # x == [3, 5, 6]
8  x = [1, None, True, ["123", 2.4]]
9  [1, 2, 3] + ["a", "b"] # [1, 2, 3, "a", "b"]
```

## Методы списка

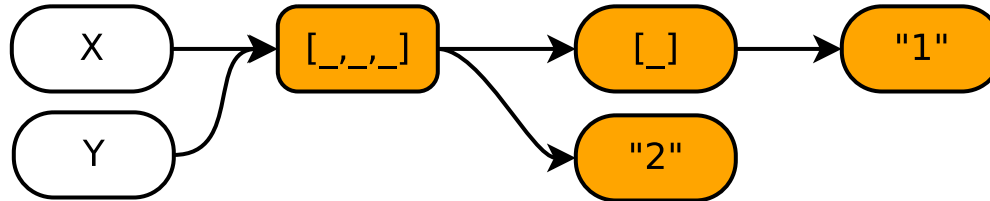
```
1  # arr.append(val)
2  [1, 2].append(3) == [1, 2, 3]
3
4  # arr.extend(arr2)
5  [1, 2].extend([2, 3]) == [1, 2, 2, 3]
6
7  # arr.pop()
8  x = [1, 2]
9  x.pop() == 2
10 print x # [1]
11
12 # arr.insert(pos, val)
13 [1, 2].insert(0, "abc") == ["abc", 1, 2]
14
15 [1, 2].index(2) == 1
16 [1, 2].reverse() == [2, 1]
17 [1, 2, 4, 1, 2, 4, 1, 1].count(1) == 4
18 x = [1, 3, 2]
19 x.remove(1) # x == [3, 2]
20 x.sort() # x == [2, 3]
```

## Изменяемые типы (ссылочные)

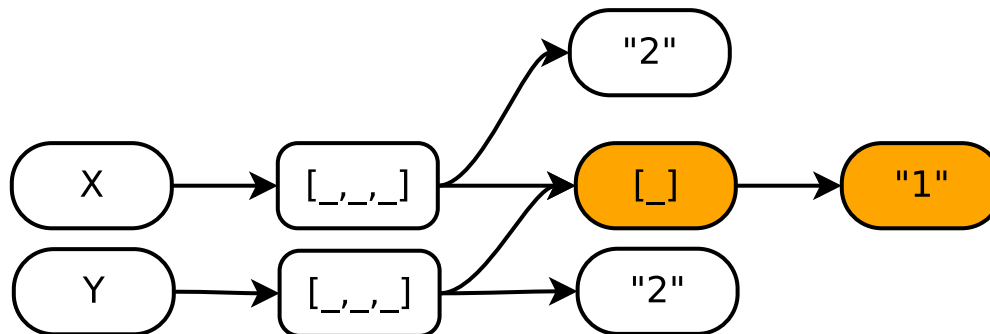
```
1     x = [1, 2, 3]
2     y = x
3     y.append(1)
4     print x => [1, 2, 3, 1]
```

## Изменяемые типы (ссылочные)

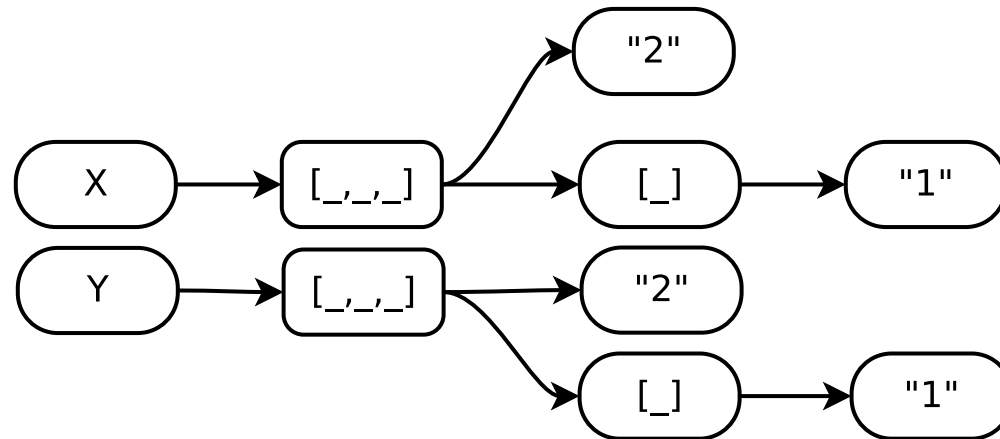
```
1 x = ["1"], "2"  
2 y = x
```



```
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
- `sort`
- Не надо сортировать неоднородные контейнеры типа - результат не определен По большей части - получится что-то, но иногда - `['x', '\xf0', u'x']`. `sort()` - `UnicodeDecodeError`

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 # error
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  $x = \{1:2, "3":4\}$
- Только константные ключи (tuple - ok)
- Элементы неупорядоченны
- Нет срезов

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

## dict – Словарь

```
1  x = {1:2, "3":"4"}
2  dict(a=1, b=2) == {"a":1, "b":2}
3  x.items() == [(1, 2), ("3", "4")]
4  x.values() == [2, "4"]
5  x.keys() == [1, "3"]
6  x.copy() == {1:2, "3":"4"}
7  x.setdefault(key, val) == val # if key not in x else x[key]
8  x.get(5, None) == None # if 5 not in x else x[5]
9  x.clear() # {}
10 x.update(y)
11 dict.fromkeys(keys, val) # {key[0]:val, key[1]:val, ..} default val is No
```

## 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

- list, tuple, set
- len длинна
- str.join `" ,␣".join ([ "1", "2", "3" ]) == "1,␣2,␣3"`
- in проверка включения элемента
- dict из [(key1, val1), (key2, val2), ...]

### Ассимптотика контейнеров

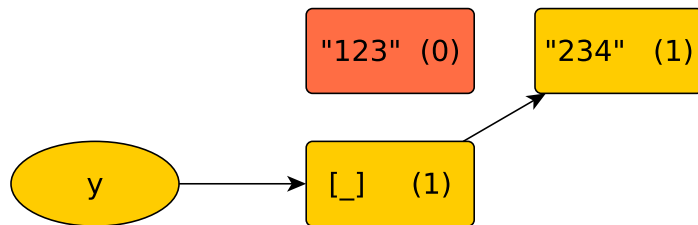
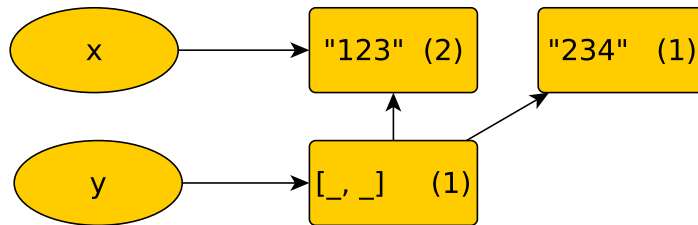
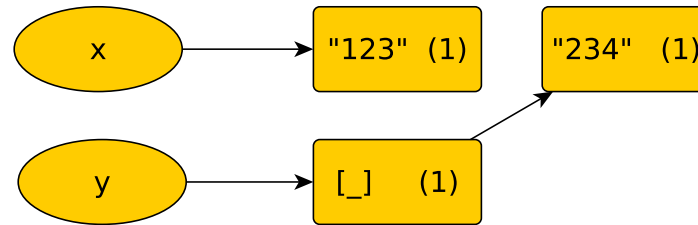
	list/tuple	dict/set	sorted list	heap
insert/remove begin	$O(n)$	$O(1)^*$	$O(n)$	$O(\log(n))$
insert/remove end	$O(1)$	$O(1)^*$	$O(n)$	$O(\log(n))$
find by key	---	$O(1)^*$	---	---
find by value	$O(n)$	$O(n)$	$O(\log(n))$	$O(n)$
find min/max	$O(n)$	$O(n)$	$O(1)^*$	$O(1)^*$



## Сборка мусора

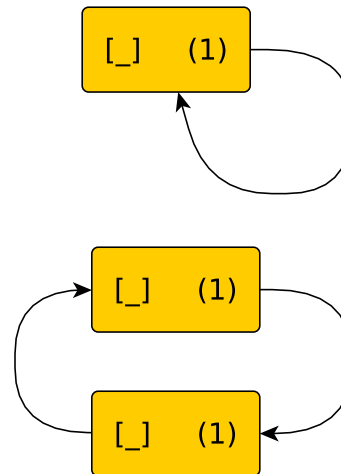
- С каждым объектом связан счетчик ссылок
- Счетчик инкрементируется при появлении каждой новой ссылки на объект - переменная, помещенная в контейнер, etc
- И декрементируется при каждом исчезновении ссылки
- Как только СС оказывается равен нулю - объект удаляется

## 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`

## Файлы

```
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"
```

## Файлы

Можно итерировать по строкам:

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