



Московский авиационный институт
(национальный исследовательский университет)

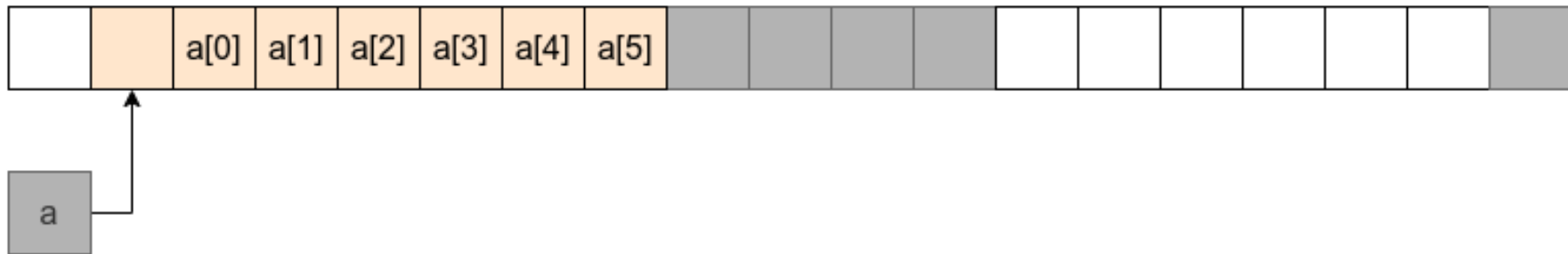
Основные структуры данных (коллекции) в Python

Коллекции в Python

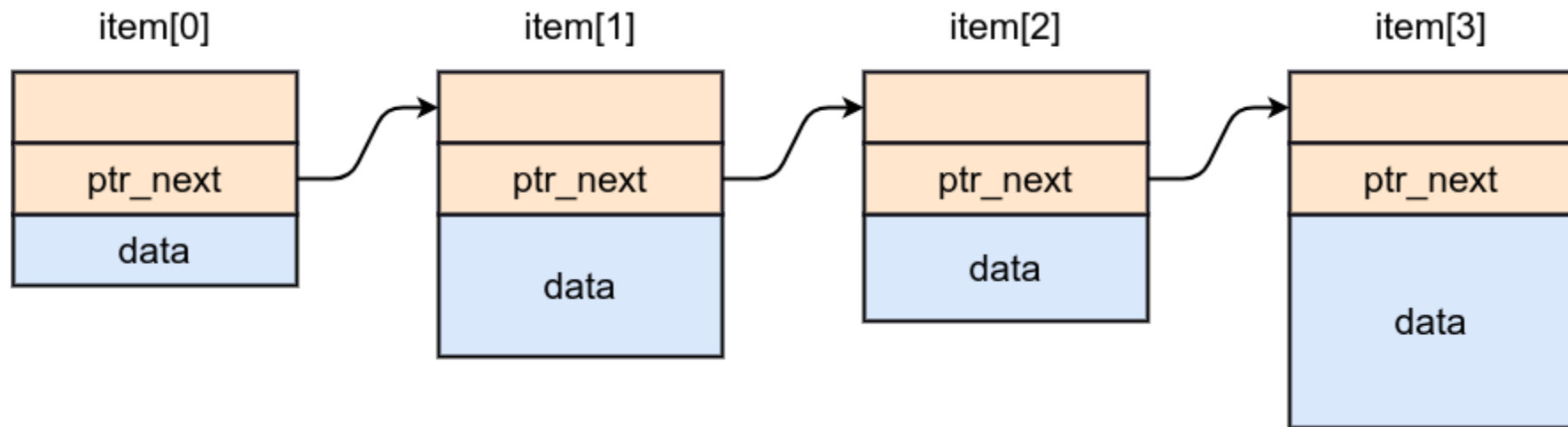
- Списки (класс `list`)
- Кортежи (класс `tuple`)
- Массивы (класс `array.array`)
- Словари (класс `dict`)
- Множества (класс `set`)
- Именованные кортежи (класс `collections.namedtuple`)
- Двусторонняя очередь (класс `collections.deque`)

Массивы и списки

Хранение данных в массиве



Классические списки



Добавление элемента в массив

1)



2)



3)

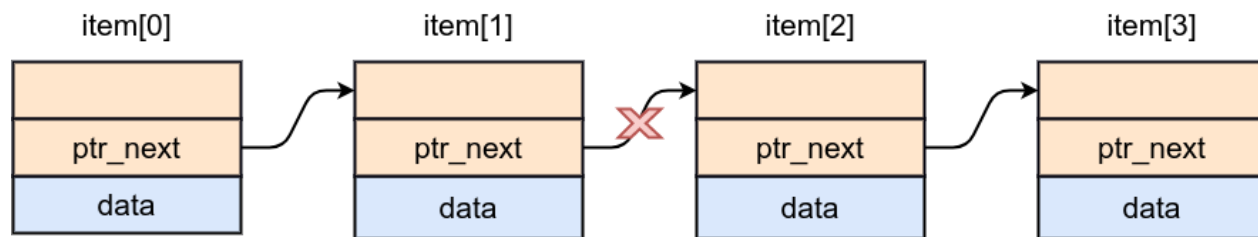


4)

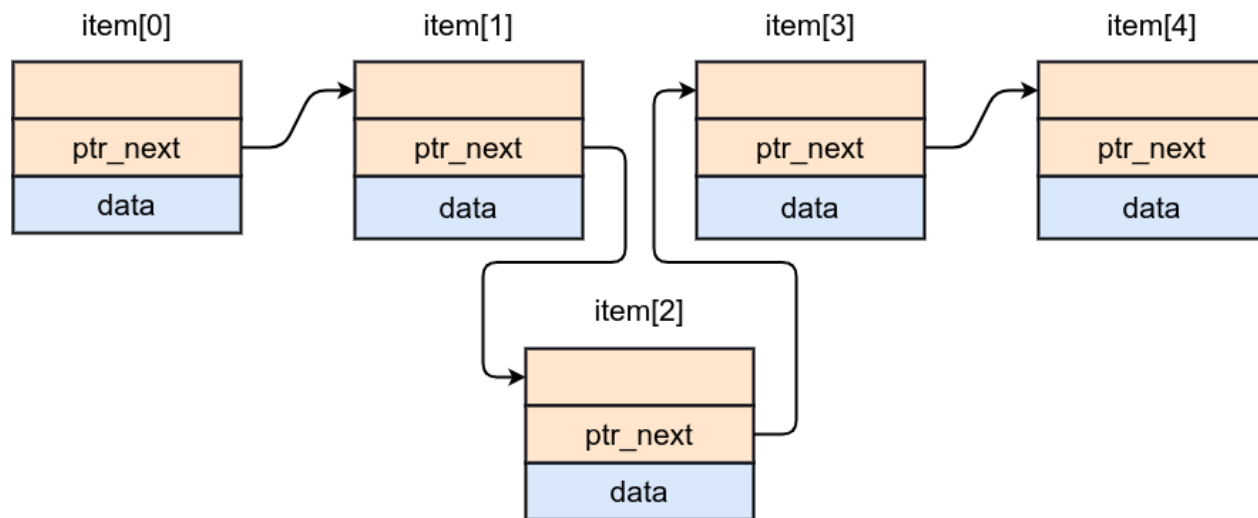


Добавление элемента классический список

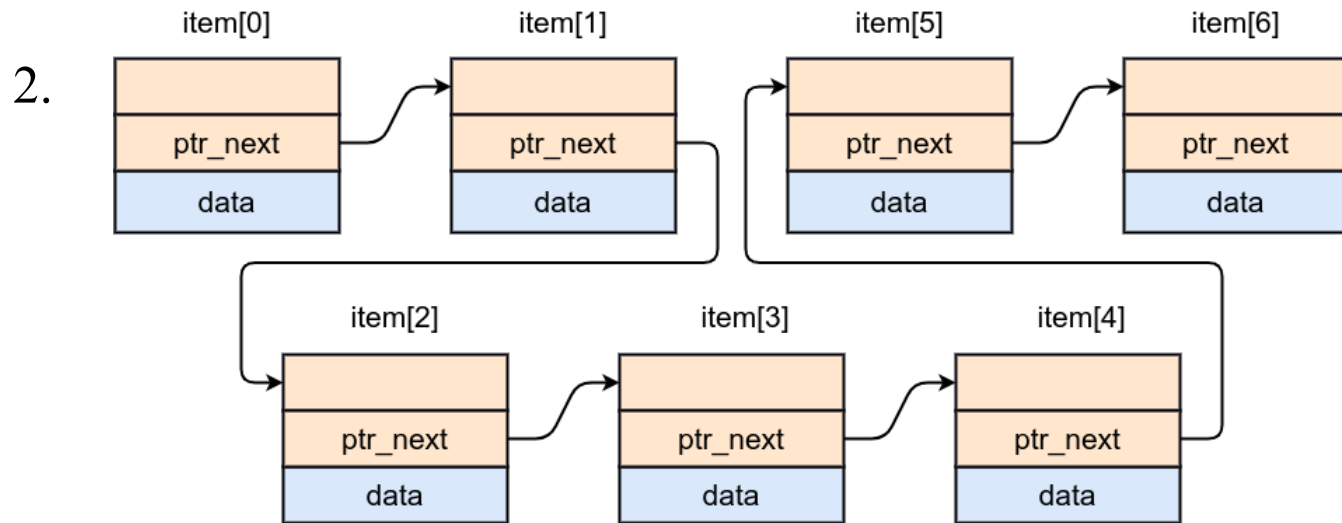
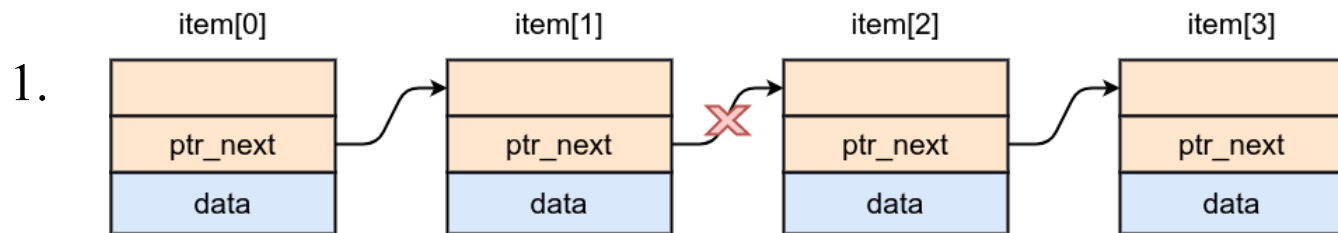
1.



2.

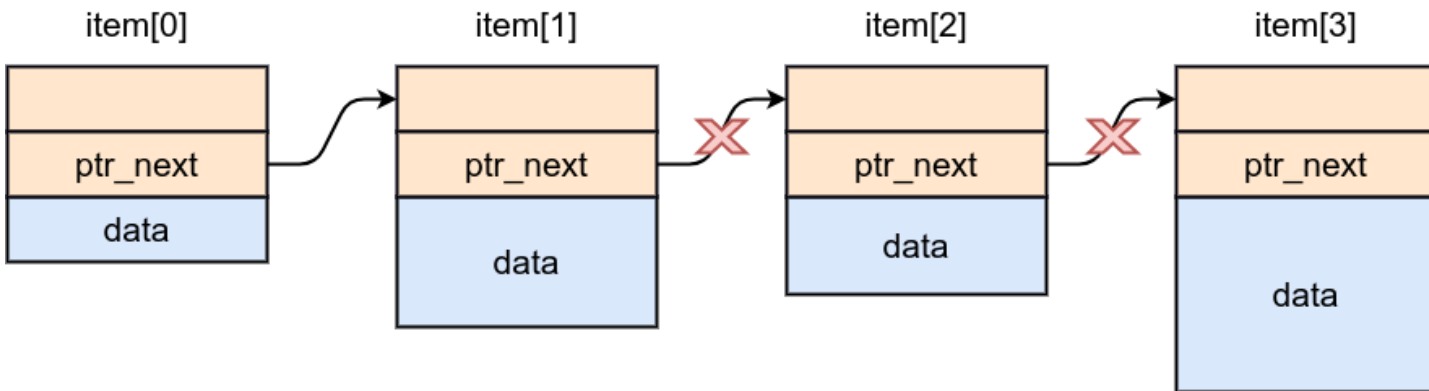


Добавление элементов в классический СПИСОК

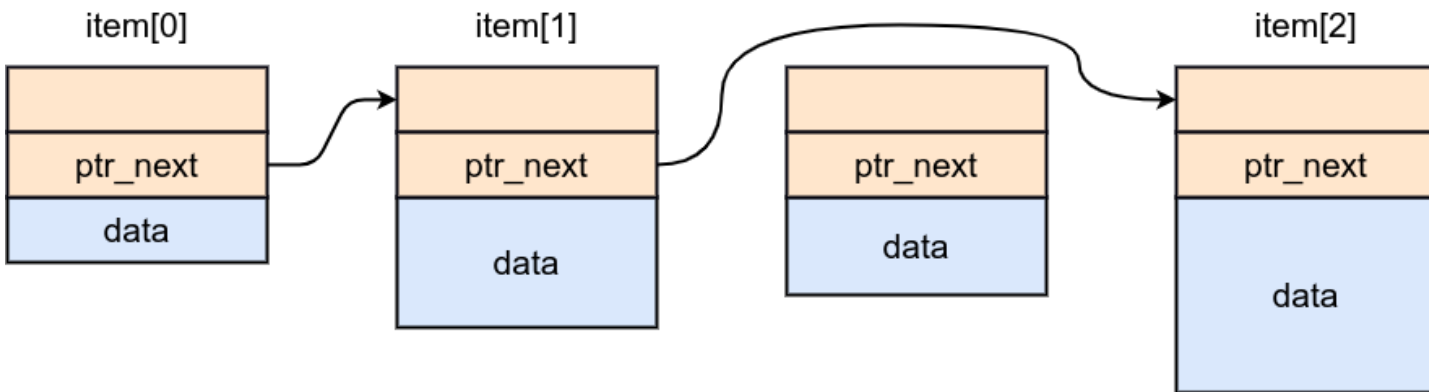


Удаление элемента из классического списка

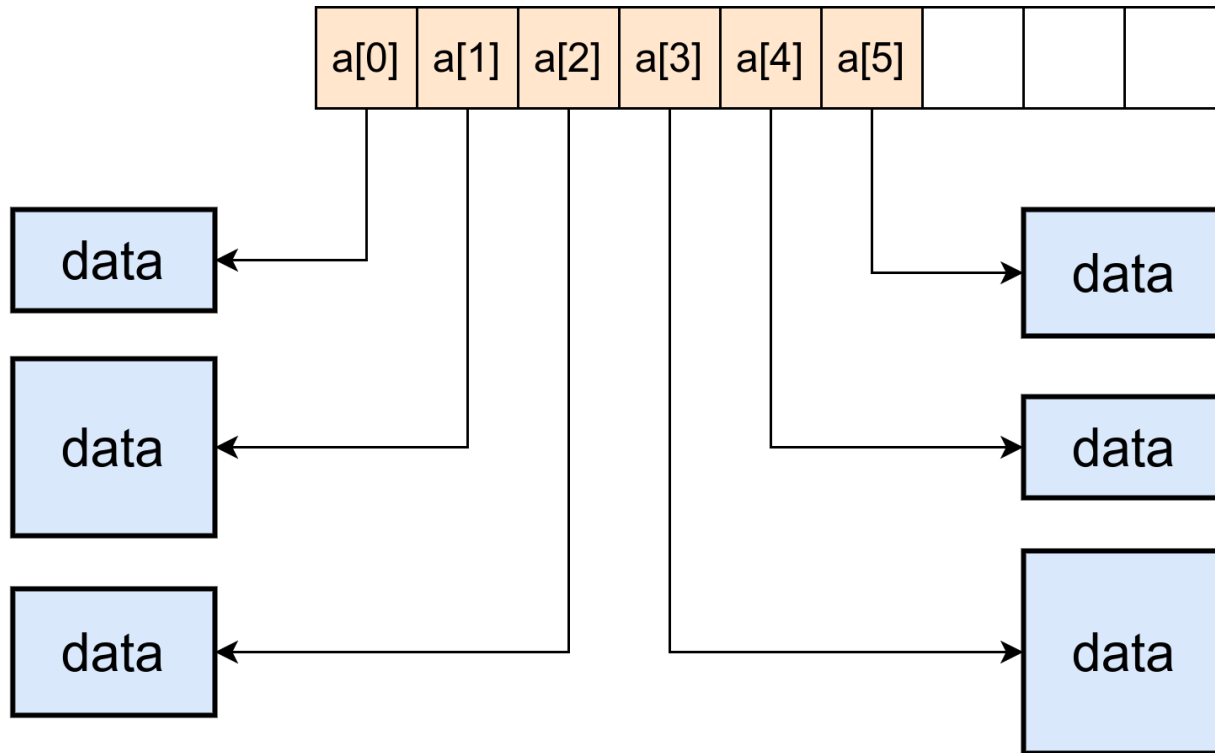
1.



2.



Списки в Python



	Список	Массив
Тип хранимых данных	Разнородные	Однородные
Требования к оперативной памяти	Больше	Меньше
Получение элемента по индексу	Медленно	Быстро
Перебор всех элементов	Быстро	Быстро
Добавление элементов	Быстро	Медленно
Удаление элементов	Быстро	Медленно

Списки (*list*)

Создание списков

```
x = []
```

```
x = list()
```

```
x = [item, item, ...]
```

```
x = [выражение с item for item in последовательность]
```

```
x = [выражение с item for item in последовательность if условие с item]
```

```
foo = [10, 20, 35, -5, 42, 16]
print("foo:", foo)
print("type(foo):", type(foo))
print()
```

```
bar = [10, 1.5, 5+2j, 'Строка']
print("bar:", bar)
```

```
foo: [10, 20, 35, -5, 42, 16]
type(foo): <class 'list'>
```

```
bar: [10, 1.5, (5+2j), 'Строка']
```

Создание пустых списков

```
foo = []  
bar = list()
```

```
print(foo)  
print(bar)
```

```
[]  
[]
```

Перенос строк при создании списков

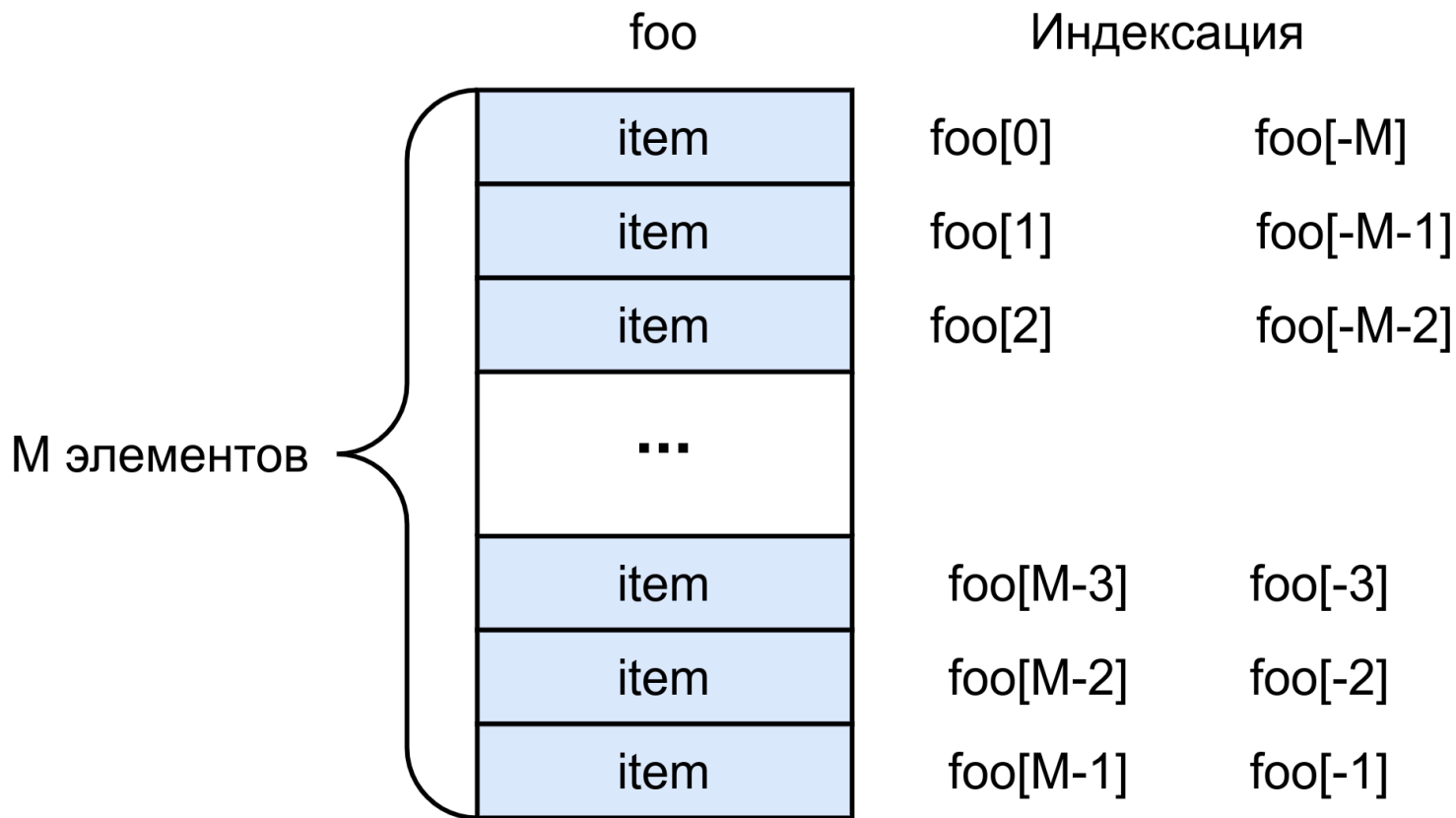
```
foo = [  
    "Lorem ipsum",  
    "dolor sit amet",  
    "consectetur adipiscing elit",  
    "sed do eiusmod",  
    "tempor incididunt"  
]
```

```
bar = [  
    "Lorem ipsum",  
    "dolor sit amet",  
    "consectetur adipiscing elit",  
    "sed do eiusmod",  
    "tempor incididunt",  
]
```


Определение длины списка (количества элементов)

```
items = [10, 20, 35, -5]  
x = len(items)
```

Индексация элементов



```
foo = [10, 20, 35, -5, 42, 16]  
print("foo:", foo)
```

```
print(foo[0])  
print(foo[1])  
print(foo[-1])  
print(foo[-2])
```

```
foo: [10, 20, 35, -5, 42, 16]  
10  
20  
16  
42
```

```
foo = [10, 20, 35, -5, 42, 16]  
print("foo:", foo)
```

```
foo[1] = 100  
foo[-2] = 200  
print("foo:", foo)
```

```
foo: [10, 20, 35, -5, 42, 16]  
foo: [10, 100, 35, -5, 200, 16]
```

Особенности работы с объектами в памяти


```
items = [1, 2, 3, 4, 5]
items_2 = items

print("items:  ", items)
print("items_2:", items_2)

print("----")

items_2[1] = 0

print("items_2:", items_2)
print("items:  ", items)
```

```
items:  [1, 2, 3, 4, 5]
items_2: [1, 2, 3, 4, 5]
----
items_2: 
items:
```

```
items = [1, 2, 3, 4, 5]
items_2 = items

print("items:  ", items)
print("items_2:", items_2)

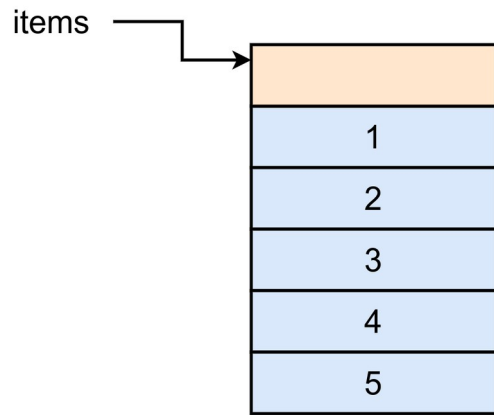
print("----")

items_2[1] = 0

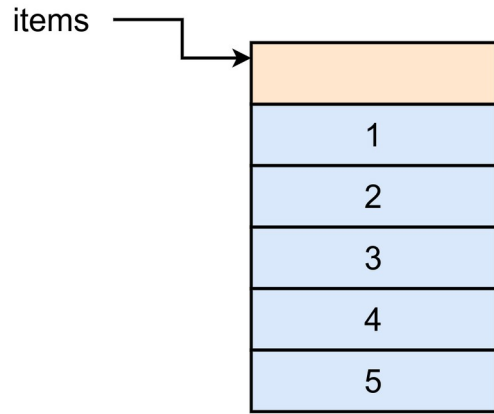
print("items_2:", items_2)
print("items:  ", items)
```

```
items:    [1, 2, 3, 4, 5]
items_2:  [1, 2, 3, 4, 5]
----
items_2:  [1, 0, 3, 4, 5]
items:    [1, 0, 3, 4, 5]
```

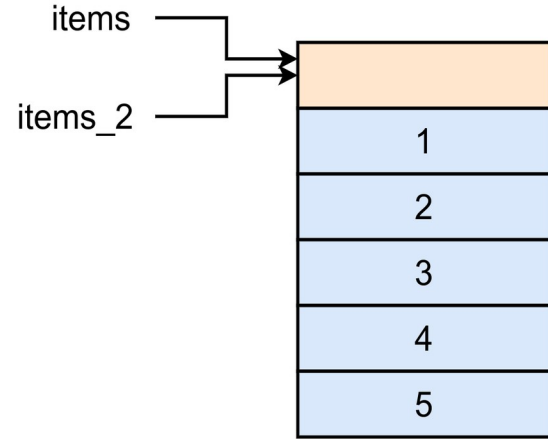
1) items = [1, 2, 3, 4, 5]



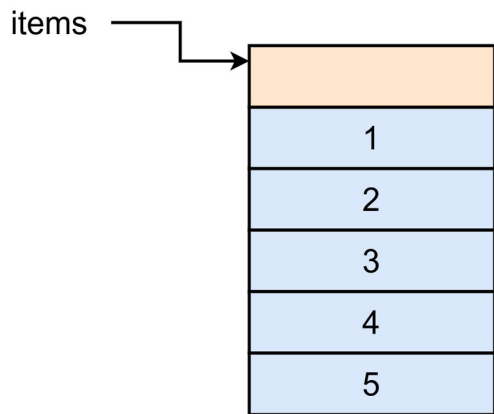
1) `items = [1, 2, 3, 4, 5]`



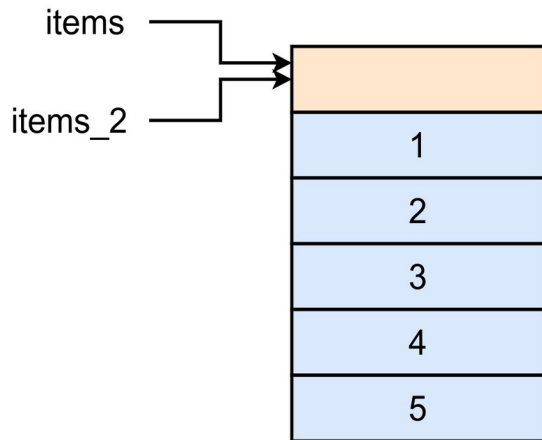
2) `items_2 = items`



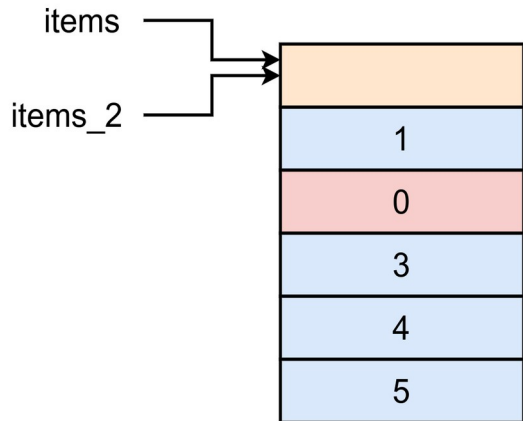
1) `items = [1, 2, 3, 4, 5]`



2) `items_2 = items`



3) `items_2[1] = 0`



`items:` [1, 2, 3, 4, 5]

`items_2:` [1, 2, 3, 4, 5]

`items_2:` [1, 0, 3, 4, 5]

`items:` [1, 0, 3, 4, 5]

```
# Ho!
```

```
x = 100
```

```
y = x
```

```
print("x:", x)
```

```
print("y:", y)
```

```
print("----")
```

```
y = 0
```

```
print("y:", y)
```

```
print("x:", x)
```

```
x: 100
```

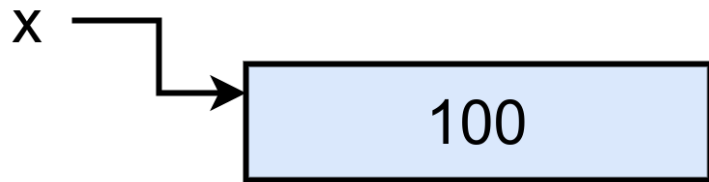
```
y: 100
```

```
----
```

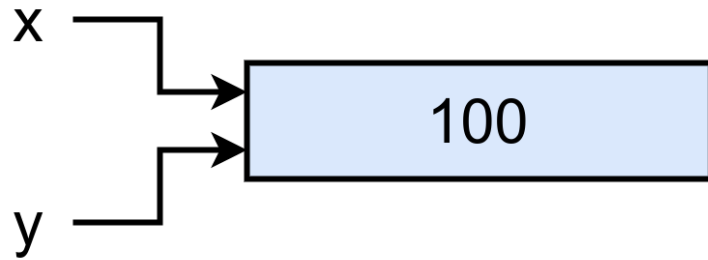
```
y: 0
```

```
x: 100
```

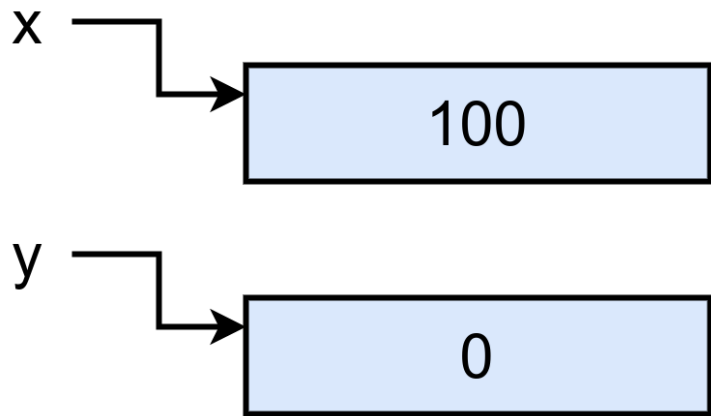
1) $x = 100$



2) $y = x$



3) $y = 0$



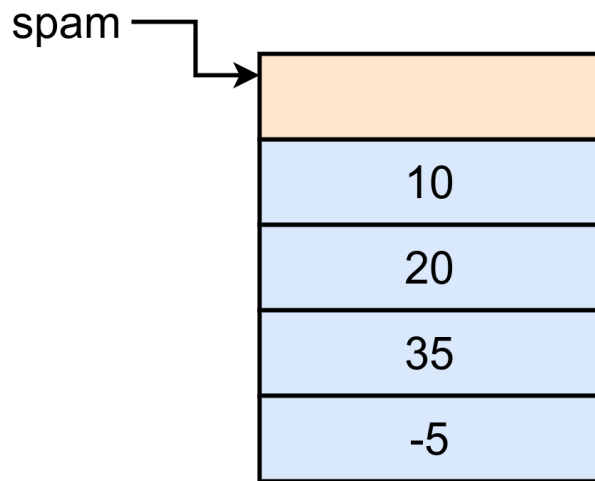
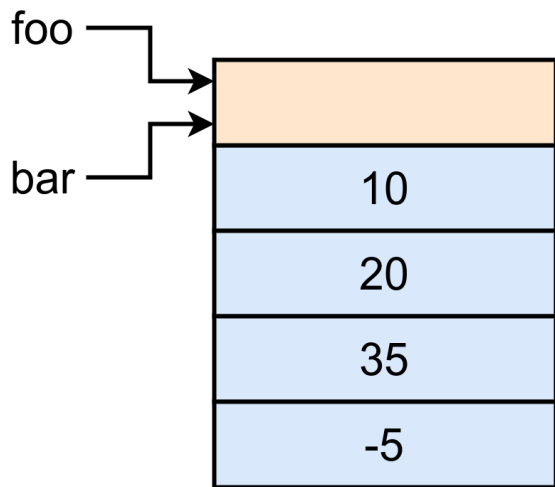
```
x: 100
y: 100
---
y: 0
x: 100
```

Операторы is / is not

```
foo = [10, 20, 35, -5]  
bar = foo  
spam = [10, 20, 35, -5]
```

```
foo == bar: True  
foo is bar: True  
foo is not bar: False
```

```
foo == spam: True  
foo is spam: False  
foo is not spam: True
```



```
foo == bar: True  
foo is bar: True  
foo is not bar: False
```

```
foo == spam: True  
foo is spam: False  
foo is not spam: True
```

Interning

Interning - это кэширование неизменяемых объектов в памяти для того, чтобы не создавать несколько экземпляров этих объектов. Это поведение может меняться от версии к версии Python.

В Python 3.11 – 3.13 заранее создаются и кэшируются объекты, представляющие собой целые числа в диапазоне $[-5; 256]$.

Так же под interning попадают строки длиной до 4096 символов включительно.

```
x = 0
y = 0
print("x:", x, "; y:", y)
print("x is y:", x is y)
```

```
x: 0 ; y: 0
x is y: True
```

```
x = 256
y = 256
print("x:", x, "; y:", y)
print("x is y:", x is y)
```

```
x: 256 ; y: 256
x is y: True
```

```
x = -5
y = -5
print("x:", x, "; y:", y)
print("x is y:", x is y)
```

```
x: -5 ; y: -5
x is y: True
```



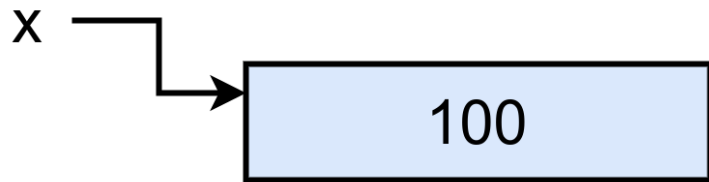
```
x = 257
y = 257
print("x:", x, "; y:", y)
print("x is y:", x is y)
```

```
x: 257 ; y: 257
x is y: False
```

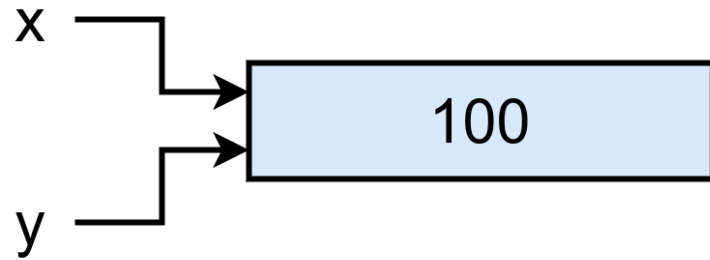
```
x = -6
y = -6
print("x:", x, "; y:", y)
print("x is y:", x is y)
```

```
x: -6 ; y: -6
x is y: False
```

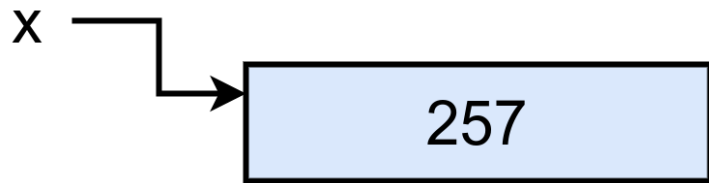
1) $x = 100$



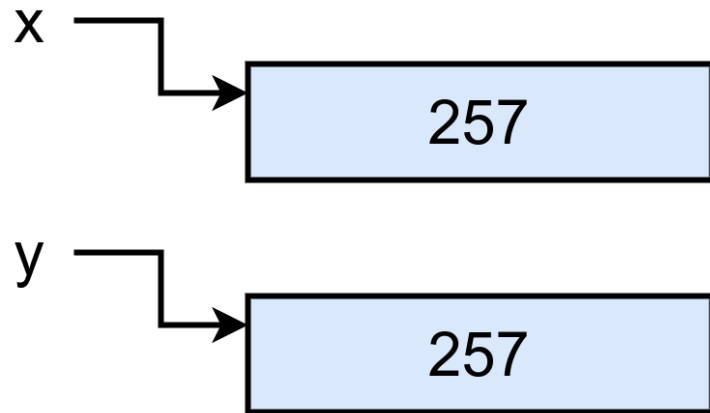
2) $y = 100$



1) $x = 257$



2) $y = 257$



```
foo = "S" * 4096
bar = "S" * 4096
print("len(foo):", len(foo), "; len(bar):", len(bar))
print("foo == bar:", foo == bar)
print("foo is bar:", foo is bar)
print()
```

```
foo = "S" * 4097
bar = "S" * 4097
print("len(foo):", len(foo), "; len(bar):", len(bar))
print("foo == bar:", foo == bar)
print("foo is bar:", foo is bar)
```

```
len(foo): 4096 ; len(bar): 4096
foo == bar: True
foo is bar: True
```

```
len(foo): 4097 ; len(bar): 4097
foo == bar: True
foo is bar: False
```

Объект *None*

```
foo = None
```

```
bar = []
```

```
print("foo is None:", foo is None)
```

```
print("bar is not None:", bar is not None)
```

```
print("type(None):", type(None))
```

```
foo is None: True
```

```
bar is not None: True
```

```
type(None): <class 'NoneType'>
```

Возвращаемся к спискам

Операторы in / not in

```
foo = [10, 20, 35, -5]
```

```
print("20 in foo:", 20 in foo)
```

```
print("42 in foo:", 42 in foo)
```

```
print("15 not in foo:", 15 not in foo)
```

```
print("-5 not in foo:", -5 not in foo)
```

```
20 in foo: True
```

```
42 in foo: False
```

```
15 not in foo: True
```

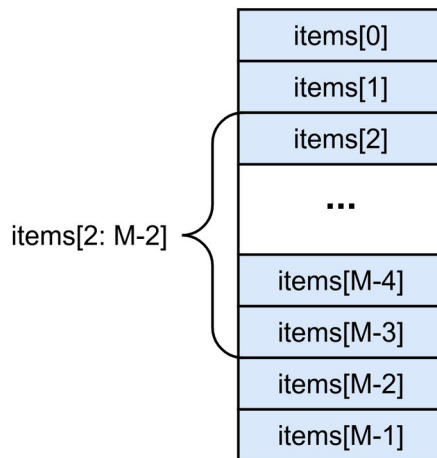
```
-5 not in foo: False
```

Срезы

$x[\text{первый элемент} : \text{последний элемент} : \text{шаг}]$

Будут получены элементы на интервале:

$[\text{первый элемент}, \text{последний элемент})$



Срезы

$x[\text{первый элемент} : \text{последний элемент} : \text{шаг}]$

Можно опустить выражения:

- *первый элемент*, если он равен 0;
- *последний элемент*, если он равен длине последовательности;
- *шаг*, если он равен 1.


```
foo = [5, 10, 12, 0, 17, 22, 4, 33, 25, 50]  
print("foo:", foo)
```

Выделение среза списка

Не включается элемент с последним номером

```
bar = foo[2: 5: 1]  
print("bar:", bar)
```

Если шаг равен 1, то его можно не указывать

```
bar = foo[2: 5]  
print("bar:", bar)
```

```
foo: [5, 10, 12, 0, 17, 22, 4, 33, 25, 50]
```

```
bar: [12, 0, 17]
```

```
bar: [12, 0, 17]
```

```
foo = [5, 10, 12, 0, 17, 22, 4, 33, 25, 50]  
print("foo:", foo)
```

Выделение среза списка с указанным шагом

```
bar = foo[2: 9: 2]  
print("bar:", bar)
```

```
foo: [5, 10, 12, 0, 17, 22, 4, 33, 25, 50]  
bar: [12, 17, 4, 25]
```

```
foo = [5, 10, 12, 0, 17, 22, 4, 33, 25, 50]
print("foo:", foo)
```

Выделение среза до конца списка

```
bar = foo[2: len(foo)]
print("bar:", bar)
```

Можно опускать правую границу,

если она равна количеству элементов в списке

```
bar = foo[2:]
print("bar:", bar)
```

```
foo: [5, 10, 12, 0, 17, 22, 4, 33, 25, 50]
```

```
bar: [12, 0, 17, 22, 4, 33, 25, 50]
```

```
bar: [12, 0, 17, 22, 4, 33, 25, 50]
```

```
foo = [5, 10, 12, 0, 17, 22, 4, 33, 25, 50]
print("foo:", foo)
```

Выделение среза от начала списка

```
bar = foo[0:5]
print("bar:", bar)
```

Можно опускать левую границу, если она равна 0

```
bar = foo[:5]
print("bar:", bar)
```

```
foo: [5, 10, 12, 0, 17, 22, 4, 33, 25, 50]
bar: [5, 10, 12, 0, 17]
bar: [5, 10, 12, 0, 17]
```

```
foo = [5, 10, 12, 0, 17, 22, 4, 33, 25, 50]  
print("foo:", foo)
```

Выделение среза от начала списка

```
bar = foo[:-2]  
print("bar:", bar)
```

```
foo: [5, 10, 12, 0, 17, 22, 4, 33, 25, 50]  
bar: [5, 10, 12, 0, 17, 22, 4, 33]
```

```
foo = [5, 10, 12, 0, 17, 22, 4, 33, 25, 50]  
print("foo:", foo)
```

Инвертирование списка

```
bar = foo[-1::-1]  
print("bar:", bar)
```

```
foo: [5, 10, 12, 0, 17, 22, 4, 33, 25, 50]  
bar: [50, 25, 33, 4, 22, 17, 0, 12, 10, 5]
```

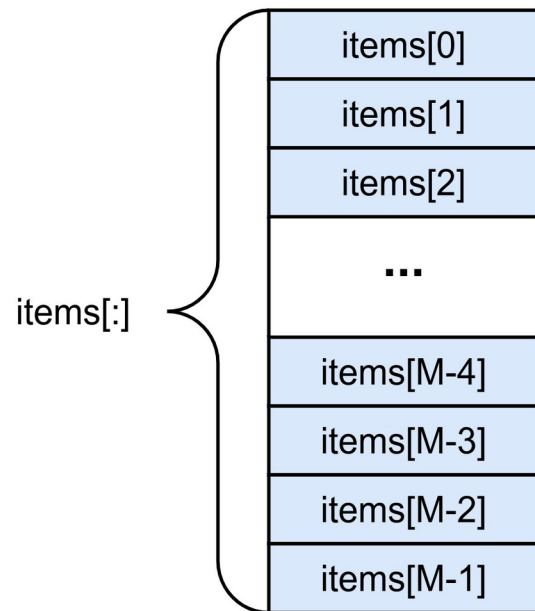
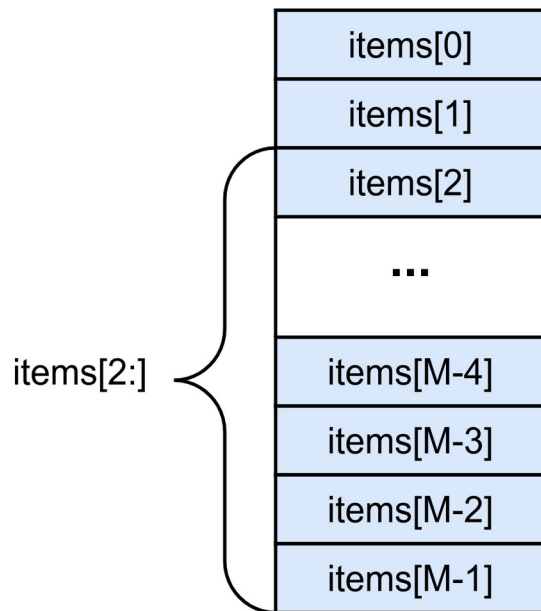
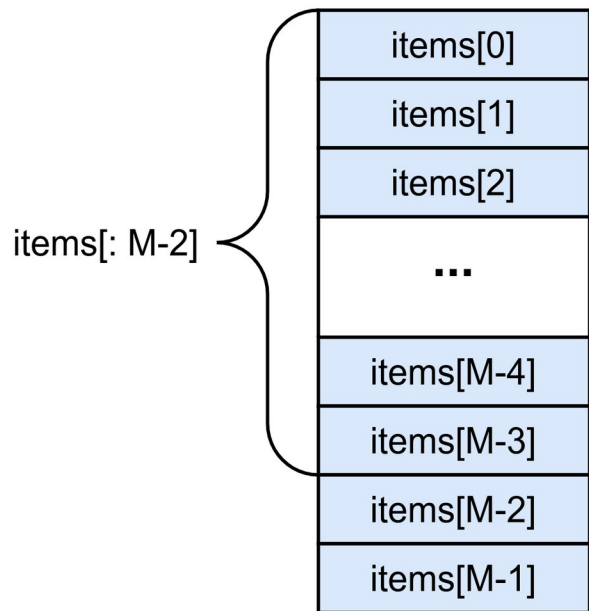
```
foo = [5, 10, 12, 0, 17, 22, 4, 33, 25, 50]  
print("foo:", foo)
```

Копирование списка

```
bar = foo[0: len(foo): 1]  
bar = foo[0: len(foo)]  
bar = foo[0:]  
bar = foo[:]  
bar = foo[::]
```

```
print("bar:", bar)
```

```
foo: [5, 10, 12, 0, 17, 22, 4, 33, 25, 50]  
bar: [5, 10, 12, 0, 17, 22, 4, 33, 25, 50]
```




```
foo = [1, 2, 3, 4, 5]
bar = foo[1:4]

print("foo:", foo)
print("bar:", bar)
print("----")

bar[0] = 0

print("bar:", bar)
print("foo:", foo)
```

```
foo: [1, 2, 3, 4, 5]
bar: [2, 3, 4]
----
bar: [0, 3, 4]
foo: [1, 2, 3, 4, 5]
```

```
foo = [1, 2, 3, 4, 5]
bar = foo[:]
```

```
print("foo:", foo)
print("bar:", bar)
print("----")
```

```
bar[0] = 0
```

```
print("bar:", bar)
print("foo:", foo)
```

```
foo: [1, 2, 3, 4, 5]
bar: [1, 2, 3, 4, 5]
----
bar: [0, 2, 3, 4, 5]
foo: [1, 2, 3, 4, 5]
```

```
foo = [1, 2, 3, 4, 5]  
bar = foo.copy()
```

```
print("foo:", foo)  
print("bar:", bar)  
print("----")
```

```
bar[0] = 0
```

```
print("bar:", bar)  
print("foo:", foo)
```

```
foo: [1, 2, 3, 4, 5]  
bar: [1, 2, 3, 4, 5]  
----  
bar: [0, 2, 3, 4, 5]  
foo: [1, 2, 3, 4, 5]
```

Методы класса *list*

```
print(dir(list))
```

```
['__add__', '__class__', '__class_getitem__', '__contains__',  
 '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__',  
 '__format__', '__ge__', '__getattr__', '__getitem__',  
 '__getstate__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__',  
 '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__',  
 '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',  
 '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__',  
 '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend',  
 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

Методы класса *list*

<code>append</code>	Добавить элемент в конец списка
<code>extend</code>	Добавить несколько элементов в список
<code>insert</code>	Добавить элемент в указанную позицию
<code>remove</code>	Удалить элемент из списка
<code>pop</code>	Удалить элемент в заданной позиции и вернуть его
<code>clear</code>	Очистить список
<code>index</code>	Найти элемент в списке
<code>count</code>	Возвращает количество заданных элементов в списке
<code>sort</code>	Сортировать список
<code>reverse</code>	Инвертировать список
<code>copy</code>	Сделать копию списка

`list.append(x)`

Add an item to the end of the list. Equivalent to `a[len(a):] = [x]`.

```
foo = [10, 20, 35, -5, 42, 16]
foo.append(100)
foo.append("text")
print("foo:", foo)
```

```
foo: [10, 20, 35, -5, 42, 16, 100, 'text']
```

```
list.extend(iterable)
```

Extend the list by appending all the items from the iterable. Equivalent to `a[len(a):] = iterable`.

```
foo = [10, 20, 35, -5, 42, 16]  
foo.extend([100, "text"])  
print("foo:", foo)
```

```
foo: [10, 20, 35, -5, 42, 16, 100, 'text']
```

`list.insert(i, x)`

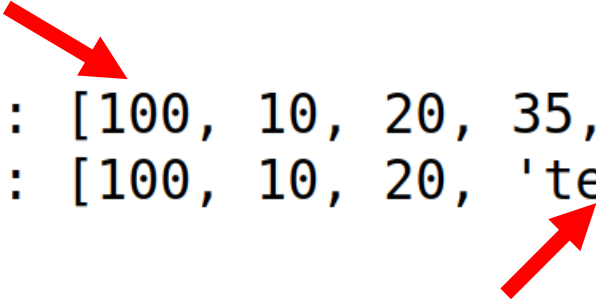
Insert an item at a given position. The first argument is the index of the element before which to insert, so `a.insert(0, x)` inserts at the front of the list, and `a.insert(len(a), x)` is equivalent to `a.append(x)`.

```
foo = [10, 20, 35, -5, 42, 16]
foo.insert(0, 100)
print("1) foo:", foo)
```

```
foo.insert(3, "text")
print("2) foo:", foo)
```

1) foo: [100, 10, 20, 35, -5, 42, 16]

2) foo: [100, 10, 20, 'text', 35, -5, 42, 16]



`list.remove(x)`

Remove the first item from the list whose value is equal to x. It raises a [ValueError](#) if there is no such item.

```
foo = [10, 20, 35, -5, 42, 42, 42, 16]
foo.remove(42)
print("1) foo:", foo)
```

```
foo.remove(10)
print("2) foo:", foo)
```

1) foo: [10, 20, 35, -5, 42, 42, 16]

2) foo: [20, 35, -5, 42, 42, 16]

```
list.remove(x)
```

Remove the first item from the list whose value is equal to x. It raises a [ValueError](#) if there is no such item.

```
foo = [10, 20, 35, -5, 42, 42, 42, 16]  
foo.remove(1000)
```

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
ValueError: list.remove(x): x not in list
```

`list.pop([i])`

Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list. (The square brackets around the *i* in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the [Python Library Reference](#).)

```
foo = [10, 20, 35, -5, 42, 16]
a = foo.pop(2)
print("1) foo:", foo, "; a:", a)

b = foo.pop()
print("2) foo:", foo, "      ; b:", b)
```

```
1) foo: [10, 20, -5, 42, 16] ; a: 35
2) foo: [10, 20, -5, 42]    ; b: 16
```

Оператор del

```
foo = [10, 20, 35, -5, 42, 16, 50, 25]
```

```
del foo[2]
```

```
print("1) foo:", foo)
```

```
del foo[-3:]
```

```
print("2) foo:", foo)
```

```
del foo[:2]
```

```
print("3) foo:", foo)
```

1) foo: [10, 20, -5, 42, 16, 50, 25]

2) foo: [10, 20, -5, 42]

3) foo: [-5, 42]

```
list.clear()
```

Remove all items from the list. Equivalent to `del a[:]`.

```
foo = [10, 20, 35, -5, 42, 16]
```

```
foo.clear()
```

```
print("foo:", foo)
```

```
foo: []
```

```
foo = [10, 20, 35, -5, 42, 16]  
bar = foo  
foo.clear()  
print("bar:", bar)
```

```
bar: []
```

```
foo = [10, 20, 35, -5, 42, 16]  
bar = foo  
foo = []  
print("bar:", bar)
```

```
bar: [10, 20, 35, -5, 42, 16]
```

```
list.index(x[, start[, end]])
```

Return zero-based index in the list of the first item whose value is equal to *x*. Raises a `ValueError` if there is no such item.

The optional arguments *start* and *end* are interpreted as in the slice notation and are used to limit the search to a particular subsequence of the list. The returned index is computed relative to the beginning of the full sequence rather than the *start* argument.

```
foo = [10, 20, 35, -5, 42, 16, 42, 20, 50, 42]
```

```
n = 42
```

```
index_1 = foo.index(n)
```

```
index_2 = foo.index(n, index_1 + 1)
```

```
print("index_1:", index_1)
```

```
print("index_2:", index_2)
```

```
index_1: 4
```

```
index_2: 6
```

```
list.count(x)
```

Return the number of times x appears in the list.

```
foo = [10, 20, 35, -5, 42, 16, 42, 20, 50, 42]  
count_42 = foo.count(42)  
count_100 = foo.count(100)
```

```
print("count_42:", count_42)  
print("count_100:", count_100)
```

```
count_42: 3  
count_100: 0
```


Оператор for

for *переменная* **in** *последовательность*:

блок кода

else:

блок кода

```
items = [5, 10, 12, 0, 17, 22, 4, 33, 25, 50]
for x in items:
    print(x * 2, end=" ")
```

10 20 24 0 34 44 8 66 50 100

Класс `range`

```
class range(stop)
```

```
class range(start, stop[, step])
```

```
foo = range(10)
print("type(foo):", type(foo))
print("foo:", foo)
print()
```

```
bar = list(foo)
print("type(bar)", type(bar))
print("bar", bar)
```

```
type(foo): <class 'range'>
foo:      range(0, 10)
```

```
type(bar)  <class 'list'>
bar        [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
foo = list(range(2, 10))  
print("foo: ", foo)
```

```
bar = list(range(2, 10, 2))  
print("bar: ", bar)
```

```
spam = list(range(10, 2, -2))  
print("spam: ", spam)
```

```
foo:    [2, 3, 4, 5, 6, 7, 8, 9]  
bar:    [2, 4, 6, 8]  
spam:   [10, 8, 6, 4]
```

Создание списка из элементов другого списка. Способ 1

```
foo = [10, 20, 35, 42, 51, 63]
bar = []
for x in foo:
    bar.append(x + 100)

print("bar: ", bar)
```

```
bar:  [110, 120, 135, 142, 151, 163]
```

Создание списка из элементов другого списка. Способ 2 (рекомендуемый)

```
foo = [10, 20, 35, 42, 51, 63]  
bar = [x + 100 for x in foo]  
print("bar: ", bar)
```

```
bar:  [110, 120, 135, 142, 151, 163]
```

Создание списка из элементов другого списка с условием. Способ 1

```
foo = [10, 20, 35, 42, 51, 63]
bar = []
for x in foo:
    if x % 2 == 0:
        bar.append(x + 100)

print("bar: ", bar)
```

bar: [110, 120, 142]

Создание списка из элементов другого списка с условием. Способ 2 (рекомендуемый)

```
foo = [10, 20, 35, 42, 51, 63]  
bar = [x + 100 for x in foo if x % 2 == 0]  
print("bar: ", bar)
```

```
bar:  [110, 120, 142]
```

```
foo = [10, 20, 35, 42, 51, 63]
bar = [x + 100
        for x in foo
        if x % 2 == 0]
print("bar: ", bar)
```

```
bar:  [110, 120, 142]
```

Оператор while

while *условие:*

блок кода

else:

блок кода

Оператор while

Текст программы: "python language/src/04/while_01.py"

```
name = None
while name != "":
    name = input("Введите имя: ")
    # if name:
    if name != "":
        print("Привет, ", name)
```

Оператор break

Текст программы: "python language/src/04/while_02.py"

```
while True:
    name = input("Введите имя: ")
    # if not name:
    if name == "":
        break

    print("Привет, ", name)
```

Оператор continue

Текст программы: "python language/src/04/while_03.py"

```
while True:
    name = input("Введите имя: ")
    # if not name:
    if name == "":
        break
    elif name == "...":
        continue

    print("Привет, ", name)
```

```
foo = [10, 20, 35, 5, 42, 16]
for n in foo:
    if n < 0:
        break
    print(n * 2, end=' ')
else:
    print()
    print("Цикл завершен")
```

20 40 70 10 84 32

Цикл завершен

```
foo = [10, 20, 35, -5, 42, 16]
for n in foo:
    if n < 0:
        break
    print(n * 2, end=' ')
else:
    print()
    print("Цикл завершен")
```

20 40 70


```
for n in range(2, 10):  
    for x in range(2, n):  
        if n % x == 0:  
            print( n, '=', x, '*', n//x)  
            break  
    else:  
        print(n, '- простое число')
```

2 - простое число
3 - простое число
4 = 2 * 2
5 - простое число
6 = 2 * 3
7 - простое число
8 = 2 * 4
9 = 3 * 3

Задачи

Задача 1

Напишите программу, которая на вход получает два списка. Из первого списка, извлечь нечетные числа, из второго — четные.

Извлеченные элементы поместить в новый список, и вывести результат.

Задача 2

Задан список целых чисел. Напишите программу, которая проверяет, все ли числа в заданном списке уникальны.

Задача 3

Напишите программу, которая принимает два списка и выводит все элементы первого, которых нет во втором.

Задача 4

Задан список, содержащий вложенные списки с целыми числами. Напишите программу, которая вычисляет сумму всех элементов, включая вложенные списки.