



Списки

Циклы



Оглавление

- Коллекции
- Конструкция for
- Оператор break
- Оператор continue
- Конструкция while



Коллекции

1. Строка (`str`)
2. Список (`list`)
3. Кортеж (`tuple`)
4. Словарь (`dict`)
5. Множество (`set`)
6. Замороженное множество (`frozenset`)



Список(List)

- Список — это упорядоченный набор элементов, перечисленных через запятую, заключённый в квадратные скобки
- Элементы списка могут быть разных типов, в отличие от элементов массива (`array`), но, как правило, используются списки из элементов одного типа
- Список может содержать одинаковые элементы, в отличие от множества (`set`)
- Список можно изменить после создания, в отличие от кортежа (`tuple`)
- Список может содержать другие списки



Список (mutable)

Создать список можно двумя способами:

Вызывать функцию `list()`

```
lst = list()
```

Использовать квадратные скобки

```
lst = [] → Задали пустой список
```

Пример:

```
lst = list([1, 4, 5])
```

```
lst = list("hello")
```

```
lst = [1, 4, 5]
```



Элементы списка разных типов

Пример:

```
>>> lst = [10, True, [1,2], "#ffffff"]
```

```
>>> type(lst)
```

```
<class 'list'>
```



Список – изменяемый тип данных

Так как список - изменяемый тип данных, то мы можем заменить определённый элемент в нём на другой.

```
>>> mass = [4, 3, 2, 1]
```

```
>>> mass[0] = 8
```

```
>>> [8, 3, 2, 1]
```



Создание копии(клона) списка

```
>>> a = [1, 3, 5, 7]
```

```
>>> b = a[:]
```

Вопрос!

Что за оператор **[:]** ?

Второй способ:

```
>>> a = [1, 3, 5, 7]
```

```
>>> b = list(a)
```




Присвоение списка

В случае, если вы выполните простое присвоение списков друг другу, то переменной **b** будет присвоена ссылка на тот же элемент данных в памяти, на который ссылается **a**

```
>>> a = [1, 3, 5, 7]
```

```
>>> b = a
```

```
>>> b[0] = 10
```

```
>>> print(a)
```

```
[10, 3, 5, 7]
```



Сложение массивов

При сложении происходит объединение множеств массива

```
>>> a = [1, 2]
```

```
>>> b = [3, 4]
```

```
>>> c = a + b
```

```
[1, 2, 3, 4]
```



Размножение списка

Мультипликация списка происходит при умножение его на число.

```
>>> a = [1, 2]
```

```
>>> b*2
```

```
[1, 2, 1, 2]
```



Нахождение значения в списке in , not in

```
>>> a = [1,2]
```

```
>>> b = 2
```

```
>>> b in a
```

```
True
```

```
>>> 19 not in a
```

```
True
```

```
>>> 1 not in a
```

```
False
```



Получить размер списка - len()

```
>>> mass = [1, 2, 3]
```

```
>>> len(mass)
```

```
3
```

предпо^ложение

Догадка, предварительная мысль.



list

- append()
- clear()
- copy()
- count()
- extend()
- index()
- insert()
- pop()
- remove()
- reverse()
- sort()

Python List Methods



WTMatter





Метод `append(x)`

Добавление элемента в список осуществляется с помощью метода `append()`

```
>>> a = []  
>>> a.append(3)  
>>> a.append("hello")  
>>> print(a)  
[3, 'hello']
```




Метод `clear()`

Метод `clear()` удаляет все элементы из списка.

```
>>> a = [1, 2, 3, 4, 5]
```

```
>>> print(a)
```

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

```
>>> a.clear()
```

```
>>> print(a)
```

```
[]
```



Метод `copy()`

Возвращает копию списка. Эквивалентно `a[:]`

```
>>> a = [1, 7, 9]
```

```
>>> b = a.copy()
```

```
>>> print(b)
```



Метод `count(x)`

Возвращает количество вхождений элемента **x** в список.

```
>>> a=[1, 2, 2, 3, 3]
```

```
>>> print(a.count(2))
```

```
2
```



Метод `extend(L)`

Расширяет существующий список за счет добавления всех элементов из списка `L`.

Эквивалентно команде `a[len(a) :] = L`

```
>>> a = [1, 2]
>>> b = [3, 4]
>>> a.extend(b)
>>> print(a)
[1, 2, 3, 4]
```



Метод `index(x[, start[, end]])`

Возвращает индекс элемента.

```
>>> a = [1, 2, 3, 4, 5]
```

```
>>> a.index(4)
```

```
3
```



Метод `insert(i, x)`

Вставить элемент `x` в позицию `i`. Первый аргумент – индекс элемента после которого будет вставлен элемент `x`.

```
>>> a = [1, 2]
>>> a.insert(0, 5)
>>> print(a)
[5, 1, 2]
>>> a.insert(len(a), 9)
>>> print(a)
[5, 1, 2, 9]
```



Метод `list.pop([i])`

Удаляет элемент из позиции `i` и возвращает его. Если использовать метод без аргумента, то будет удален последний элемент из списка.

```
>>> a = [1, 2, 3, 4, 5]
```

```
>>> print(a.pop(2))
```

```
3
```

```
>>> print(a.pop())
```

```
5
```

```
>>> print(a)
```

```
[1, 2, 4]
```



Метод `remove(x)`

Удаляет первое вхождение элемента `x` из списка.

```
>>> a = [1, 2, 3]
```

```
>>> a.remove(1)
```

```
>>> print(a)
```

```
[2, 3]
```




Метод `reverse()`

Изменяет порядок расположения элементов в списке на обратный.

```
>>> a = [1, 3, 5, 7]
```

```
>>> a.reverse()
```

```
>>> print(a)
```

```
[7, 5, 3, 1]
```



Метод `sort()`

Сортирует элементы в списке по возрастанию. Для сортировки в обратном порядке используйте флаг `reverse=True`. Дополнительные возможности открывает параметр `key`, за более подробной информацией обратитесь к документации.

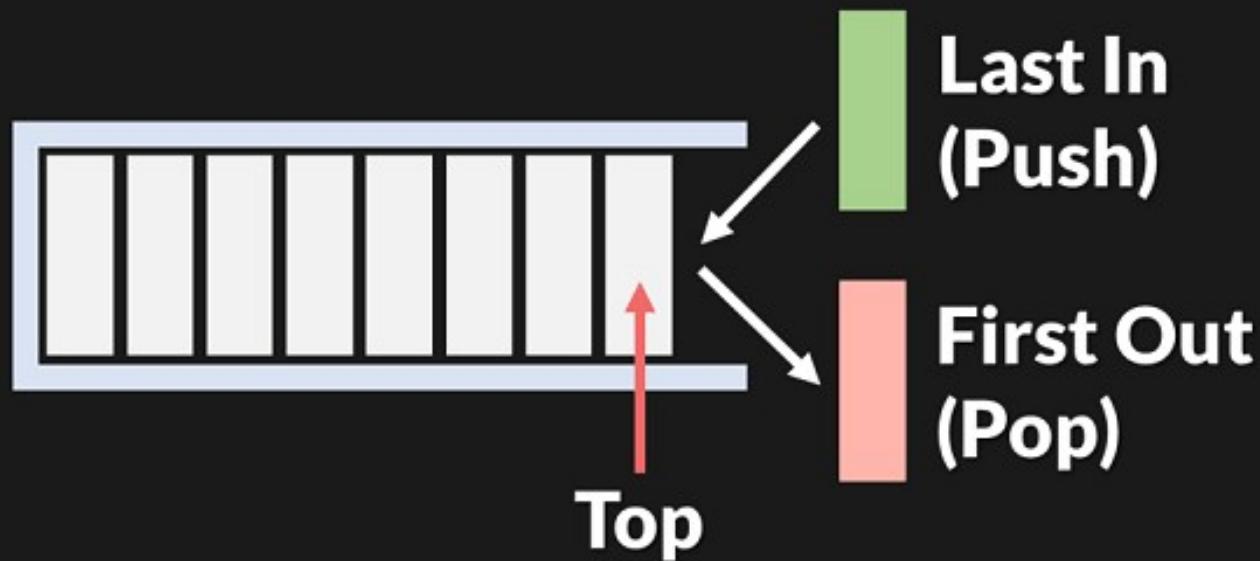
```
>>> a = [1, 4, 2, 8, 1]
>>> a.sort()
>>> print(a)
[1, 1, 2, 4, 8]
```



Как запомнить все эти методы ?



Python Stack Implementation



Срезы(слайсы) в массивах

Срез как и в строках задается тройкой чисел `[start:stop:step]`
`start` – индекс с которой нужно начать выборку, `stop` – конечный индекс, `step` – шаг. При этом необходимо помнить, что выборка не включает элемент определяемый индексом `stop`.

```
>>> # Получить копию списка
>>> a[:]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> # Получить первые пять элементов списка
>>> a[0:5]
[0, 1, 2, 3, 4]
>>> # Получить элементы с 3-го по 7-ой
>>> a[2:7]
[2, 3, 4, 5, 6]
>>> # Взять из списка элементы с шагом 2
>>> a[::2]
[0, 2, 4, 6, 8]
>>> # Взять из списка элементы со 2-го по 8-ой с шагом 2
>>> a[1:8:2]
[1, 3, 5, 7]
```



Цикл for

Общая конструкция:

for цель **in** объект:

операторы

if проверка: break # выход из цикла

if проверка: continue # переход в начало цикла

else:

Операторы # ветка else выполняется если не было выхода с
помощью оператора break



Итерация списка с использованием for

```
input_list = [10, "S", 15, "A", 1]
for x in input_list:
    print(x)
```

Вывод:

10

"S"

15

"A"

1



Функция range()

Функция `range()` применяется для генерации индексов в цикле `for`. Генерирует диапазон чисел в зависимости от условия.

```
>>> range(5)
```

```
>>> list(range(5))
```

```
[0, 1, 2, 3, 4]
```

```
>>> list(range(-5, 5))
```

```
[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4]
```




Пример:

```
for x in range(3):  
    print('result', x )
```



А можно так ?

```
for x in 3:  
    print('result', x )
```



Оператор break

```
>>> for i in 'hello world':  
...     if i == 'o':  
...         break  
...     print(i * 2, end=' ')  
...  
hheel111
```



Оператор continue

```
>>> for i in 'hello world':  
...     if i == 'o':  
...         continue  
...     print(i * 2, end=' ')  
...  
hheellll  wwrrlldd
```



Волшебное слово else

```
>>> for i in 'hello world':  
...     if i == 'a':  
...         break  
...     else:  
...         print('Буквы а в строке нет')  
...
```

Буквы а в строке нет



Оператор pass

```
for i in 'hello world':
```

```
    ????
```

← Что нужно поставить чтобы for заработал ?



Цикл while

Общая конструкция:

while проверка условия:

операторы

if проверка: break # выход из цикла

if проверка: continue # переход в начало цикла

else:

Операторы # ветка else выполняется если не было выхода с
помощью оператора break



Пример

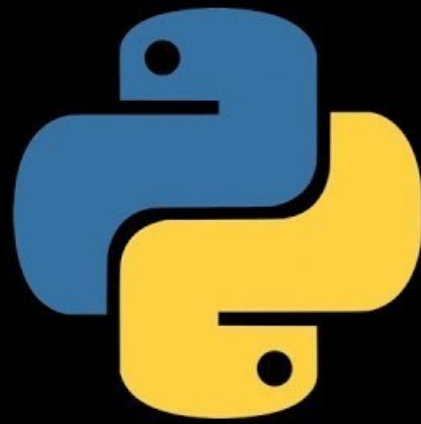
```
>>> i = 5
>>> while i < 15:
...     print(i)
...     i = i + 2
...
5
7
9
11
13
```




Бесконечный цикл

```
>>> i = 5
>>> while True:
...     print(i)
...     i = i + 2
...     if i == 7: break
```

Что выведет код ?



PYTHON

PROGRAMMING