

Использование списков list



 винилплитка.рф

РЕКЛАМА

Кварцвиниловая плитка в Красноярске!

Акции всегда • Доставка по звонку • Подъем до квартиры • Хранение бесплатно

[Узнать больше](#)

[Справочник по языку Python3.](#) / Использование списков list

Язык программирования Python имеет несколько составных типов данных, используемых для группировки значений. Наиболее универсальным является [список](#), который можно записать в виде списка значений (элементов), разделенных запятыми, в квадратных скобках. [Списки могут содержать элементы разных типов](#), но обычно все элементы имеют одинаковый тип.

```
>>> squares = [1, 4, 9, 16, 25]
>>> squares
[1, 4, 9, 16, 25]
```

Как и все другие встроенные [типы последовательностей](#), списки можно [индексировать](#) и [извлекать срезы](#):

```
# индексация возвращает элемент
>>> squares[0]
# 1
>>> squares[-1]
# 25

# срез возвращает новый список
>>> squares[-3:]
# [9, 16, 25]
```

Все [операции срезов](#) возвращают новый список, содержащий запрошенные элементы. Это означает, что следующий фрагмент возвращает поверхностную копию списка:

```
# копирование списка
>>> cp = squares[:]
>>> cp
# [1, 4, 9, 16, 25]
>>> cp.remove(1)
>>> cp
# [4, 9, 16, 25]

# список squares не изменился
>>> squares
# [1, 4, 9, 16, 25]
```

Списки также поддерживают такие операции, как конкатенация:

```
>>> squares + [36, 49, 64, 81, 100]
# [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

В отличие от [текстовых строк](#), которые являются неизменяемыми, **списки являются изменяемым типом**, то есть можно изменить их содержимое:

```
>>> cubes = [1, 8, 27, 65, 125]
>>> 4 ** 3
# 64

# заменим неправильное значение
>>> cubes[3] = 64
```

```
>>> cubes
# [1, 8, 27, 64, 125]
```

Можно добавить новые элементы в конец списка, используя [list.append\(\)](#).

```
>>> cubes.append(216)  # add the cube of 6
>>> cubes.append(7 ** 3)  # and the cube of 7
>>> cubes
# [1, 8, 27, 64, 125, 216, 343]
```

Возможно присвоение срезов, и это может даже изменить размер списка или полностью очистить его:

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> letters
# ['a', 'b', 'c', 'd', 'e', 'f', 'g']

# заменить некоторые значения
>>> letters[2:5] = ['C', 'D', 'E']
>>> letters
# ['a', 'b', 'C', 'D', 'E', 'f', 'g']

# теперь удалим их
>>> letters[2:5] = []
>>> letters
# ['a', 'b', 'f', 'g']

# очистим список, заменив все элементы пустым списком
>>> letters[:] = []
>>> letters
# []
```

Встроенная [функция len\(\)](#), которая вычисляет количество элементов в списке, также применяется к спискам:

```
>>> letters = ['a', 'b', 'c', 'd']
>>> len(letters)
# 4
```

Можно вкладывать списки (создавать списки, содержащие другие списки), например:

```
>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> x
# [['a', 'b', 'c'], [1, 2, 3]]

>>> x[0]
# ['a', 'b', 'c']

>>> x[0][1]
# 'b'
```

Дополнительно смотрите:

- Встроенный класс [list\(\)](#).
- [Тип list - списки в Python](#).
- [Простая и расширенная распаковка списков и кортежей в Python](#).
- [Общие операции с последовательностями list, tuple, str в Python](#).
- [Операции с изменяемыми последовательностями list в Python](#).

Все методы списков в Python:

Такие методы, как `list.insert`, `list.remove` или `list.sort`, которые только изменяют список, не печатают возвращаемое значение, они возвращают значение `None` по умолчанию. Это принцип проектирования для всех изменяемых структур данных в Python.

Кроме того, можно заметить, что не все данные могут быть отсортированы или сравнены. Например, `[None, 'hello', 10]` не сортируется, потому что целые числа нельзя сравнить со строками, а `None` нельзя сравнить с другими типами. Кроме того, есть некоторые типы, которые не имеют определенного упорядочения. Например выражение `3+4j < 5+7j` [комплексных чисел](#) не является допустимым сравнением.

- [list.append\(x\)](#):
Добавляет элемент в конец списка. Эквивалент `lst[len(lst):] = [x]`
- [list.extend\(iterable\)](#):
Расширяет список, добавив все элементы из [последовательности](#) которая поддерживает [итерацию](#). Эквивалент `lst[len(lst):] = iterable`
- [list.insert\(i, x\)](#):
Вставляет элемент в заданную позицию. Первый аргумент - это индекс элемента, перед которым можно вставить, поэтому `lst.insert(0, x)` вставляется в начало списка, а выражение `lst.insert(len(lst), x)` эквивалентно `a.append(x)`.
- [list.remove\(x\)](#):
Удаляет первый элемент из списка, значение которого равно `x`. Поднимает `ValueError`, если такого элемента нет.
- [list.pop\(\[i\]\)](#):
Возвращает элемент в указанной позиции и удаляет этот элемент из списка. Если индекс не указан `lst.pop()`, то удаляет и возвращает последний элемент из списка. Квадратные скобки вокруг `i` в сигнатуре метода означают, что параметр является необязательным, а не то, что нужно вводить квадратные скобки в этой позиции. Это обозначение часто видно в [Справочнике по библиотеке Python](#).
- [list.clear\(\)](#):
Удаляет все элементы из списка. Эквивалент `del a[:]`.
- [list.index\(x\[, start\[, end\]\]\)](#):
Возвращает нулевой индекс в списке первого элемента, значение которого равно `x`. Поднимает `ValueError`, если такого элемента нет.

Необязательные аргументы `start` и `end` интерпретируются так же, как в [нотации среза](#), и используются для ограничения поиска определенной подпоследовательностью списка. Возвращенный [индекс вычисляется относительно начала полной последовательности](#), а не аргумента `start`.
- [list.count\(x\)](#):
Возвращает количество появлений значения `x` в списке.
- [list.sort\(key=None, reverse=False\)](#) (см. в описании типа метод `list.sort`):
Сортировка элементов списка на месте. Аргументы могут быть использованы для настройки сортировки, значения аргументов, такие-же как во [встроенной функции sorted\(\)](#).
- [list.reverse\(\)](#):
Меняет местами элементы списка. Переворачивает список.
- [list.copy\(\)](#):
Возвращает мелкую копию списка. Эквивалент `lst[:]`.

Содержание раздела:

- [КРАТКИЙ ОБЗОР МАТЕРИАЛА.](#)
- [Список Python как аргумент по умолчанию](#)
- [Использование списка в качестве стека](#)
- [Использование списка в качестве очереди](#)
- [Генератор списка list](#)
- [Эффективное использование генераторов списков](#)
- [Операция присваивания на месте и списки](#)
- [Поведение списков Python в разных областях видимости](#)
- [Сравнение и идентичность двух списков](#)

- [Как получить несколько последних элементов списка](#)

ХОЧУ ПОМОЧЬ
ПРОЕКТУ



[DOCS-Python.ru](#)[™], 2023 г.

(Внимание! При копировании материала ссылка на источник обязательна)

[@docs_python_ru](#)