Списки в Python - упорядоченные изменяемые коллекции объектов произвольных типов (почти как массив, но типы могут отличаться).

Чтобы использовать списки, их нужно создать. Создать список можно несколькими способами. Например, можно обработать любой итерируемый объект (например, [строку](https://pythonworld.ru/tipy-dannyx-v-python/stroki-literaly-strok.html)) встроенной функцией **list**:

>>>

**>>>** list('список')

['с', 'п', 'и', 'с', 'о', 'к']

|  |
| --- |
| numbers = [1, 2, 3, 4, 5] |

Также для создания списка можно использовать конструктор **list()**:

|  |  |
| --- | --- |
| 1  2 | numbers1 = []  numbers2 = list() |

Оба этих определения списка аналогичны - они создают пустой список.

Конструктор list для создания списока может принимать другой список:

|  |  |
| --- | --- |
| 1  2 | numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]  numbers2 = list(numbers) |

Список можно создать и при помощи литерала:

>>>

**>>>** s = [] *# Пустой список*

**>>>** l = ['s', 'p', ['isok'], 2]

**>>>** s

[]

**>>>** l

['s', 'p', ['isok'], 2]

Как видно из примера, список может содержать любое количество любых объектов (в том числе и вложенные списки), или не содержать ничего.

И еще один способ создать список - это **генераторы списков**. Генератор списков - способ построить новый список, применяя выражение к каждому элементу последовательности. Генераторы списков очень похожи на цикл [for](https://pythonworld.ru/osnovy/cikly-for-i-while-operatory-break-i-continue-volshebnoe-slovo-else.html).

>>>

**>>>** c = [c \* 3 **for** c **in** 'list']

**>>>** c

['lll', 'iii', 'sss', 'ttt']

Возможна и более сложная конструкция генератора списков:

>>>

**>>>** c = [c \* 3 **for** c **in** 'list' **if** c != 'i']

**>>>** c

['lll', 'sss', 'ttt']

**>>>** c = [c + d **for** c **in** 'list' **if** c != 'i' **for** d **in** 'spam' **if** d != 'a']

**>>>** c

['ls', 'lp', 'lm', 'ss', 'sp', 'sm', 'ts', 'tp', 'tm']

Но в сложных случаях лучше пользоваться обычным циклом for для генерации списков

Для обращения к элементам списка надо использовать индексы, которые представляют номер элемента в списка. Индексы начинаются с нуля. То есть второй элемент будет иметь индекс 1. Для обращения к элементам с конца можно использовать отрицательные индексы, начиная с -1. То есть у последнего элемента будет индекс -1, у предпоследнего - -2 и так далее.

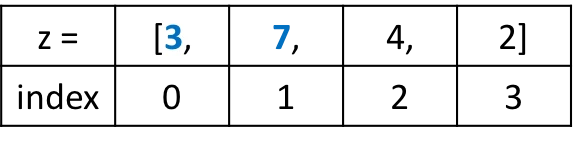
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | numbers = [1, 2, 3, 4, 5]  print(numbers[0])   # 1  print(numbers[2])   # 3  print(numbers[-3])  # 3    numbers[0] = 125  # изменяем первый элемент списка  print(numbers[0])   # 125 |

Если необходимо создать список, в котором повторяется одно и то же значение несколько раз, то можно использовать символ звездочки \*. Например, определим список из шести пятерок:

|  |  |
| --- | --- |
| 1  2 | numbers = [5] \* 6  # [5, 5, 5, 5, 5, 5]  print(numbers) |

## Срезы(slice) списка

Срезы хороши для получения подмножества значений с вашего списка. На примере кода, приведенного ниже, он вернет список с элементами из индекса 0 и не включая индекс 2.

Первый индекс пишется (до : включительно), а последний (после : ) и не учитывается

КОПИРОВАТЬ

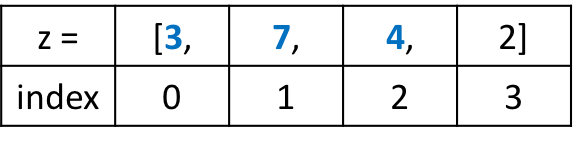
# Создайте список

z = [3, 7, 4, 2]

# Вывод элементов с индексом от 0 до 2 (не включая 2)

print(z[0:2])

# вывод: [3, 7]

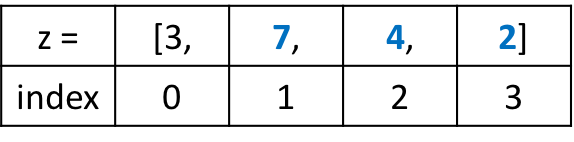


КОПИРОВАТЬ

# Все, кроме индекса 3

>>> print(z[:3])

[3, 7, 4]



Код, указанный ниже возвращает список с элементами начиная с индекса 1 до конца.

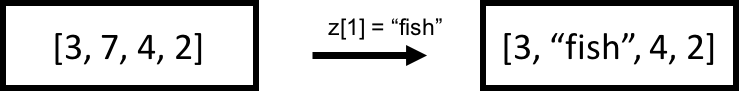
КОПИРОВАТЬ

# начиная с индекса 1 до конца списка

>>> print(z[1:])

[7, 4, 2]

## Изменение элементов в списке



[**Списки в Python**](https://pythonru.com/primery/python-spiski-primery) изменяемы. Это означает, что после создания списка можно обновить его отдельные элементы.

КОПИРОВАТЬ

z = [3, 7, 4, 2] # Создание списка

# Изменяем элемент с индексом 1 на строку 'fish'

z[1] = 'fish'

print(z)

[3, 'fish', 4, 2]

Кроме того, если нам необходим последовательный список чисел, то для его создания удобно использовать функцию **range**, которая имеет три формы:

* range(end): создается набор чисел от 0 до числа end
* range(start, end): создается набор чисел от числа start до числа end
* range(start, end, step): создается набор чисел от числа start до числа end с шагом step

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | numbers = list(range(10))  print(numbers)      # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  numbers = list(range(2, 10))  print(numbers)      # [2, 3, 4, 5, 6, 7, 8, 9]  numbers = list(range(10, 2, -2))  print(numbers)      # [10, 8, 6, 4] |

Например, следующие два определения списка будут аналогичны, но за счет функции range мы сокращаем объем кода:

|  |  |
| --- | --- |
| 1  2 | numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]  numbers2 = list(range(1, 10)) |

Список необязательно должен содержать только однотипные объекты. Мы можем поместить в один и тот же список одновременно строки, числа, объекты других типов данных:

|  |  |
| --- | --- |
| 1 | objects = [1, 2.6, "Hello", True] |

**Перебор элементов**

Для перебора элементов можно использовать как цикл for, так и цикл while.

Перебор с помощью цикла for:

|  |  |
| --- | --- |
| 1  2  3 | companies = ["Microsoft", "Google", "Oracle", "Apple"]  for item in companies:      print(item) |

Здесь вместо функции range мы сразу можем подставить имеющийся список companies.

Перебор с помощью цикла while:

|  |  |
| --- | --- |
| 1  2  3  4  5 | companies = ["Microsoft", "Google", "Oracle", "Apple"]  i = 0  while i < len(companies):      print(companies[i])      i += 1 |

Для перебора с помощью функции **len()** получаем длину списка. С помощью счетчика i выводит по элементу, пока значение счетчика не станет равно длине списка.

**Сравнение списков**

Два списка считаются равными, если они содержат один и тот же набор элементов:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]  numbers2 = list(range(1,10))  if numbers == numbers2:      print("numbers equal to numbers2")  else:      print("numbers is not equal to numbers2") |

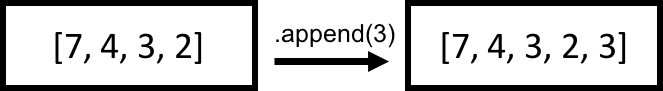
В данном случае оба списка будут равны.

### Методы и функции по работе со списками

Для управления элементами списки имеют целый ряд методов. Некоторые из них:

* **append(item)**: добавляет элемент item в конец списка

### Метод Append

Добавьте значение 3 в конец списка

Метод append добавляет элемент в конец списка. Это происходит на месте.

КОПИРОВАТЬ

z = [7, 4, 3, 2]

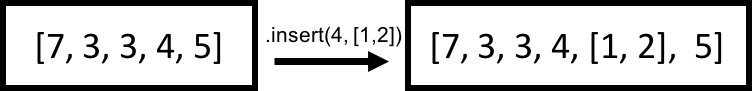
z.append(3)

print(z)

[7, 4, 3, 2, 3]

* **insert(index, item)**: добавляет элемент item в список по индексу index

### Метод Insert

Вставляет [1,2] с индексом 4

Метод insert вставляет элемент перед указанным индексом.

КОПИРОВАТЬ

z = [7, 3, 3, 4, 5]

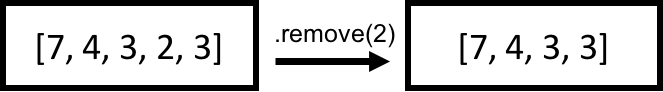
z.insert(4, [1, 2])

print(z)

[7, 3, 3, 4, [1, 2], 5]

* **remove(item)**: удаляет элемент item. Удаляется только первое вхождение элемента. Если элемент не найден, генерирует исключение ValueError

### Метод Remove

Метод remove удаляет первое вхождение значения в списке.

КОПИРОВАТЬ

z = [7, 4, 3, 2, 3]

z.remove(2)

print(z)

Код удаляет первое вхождение значения 2 из списка z.

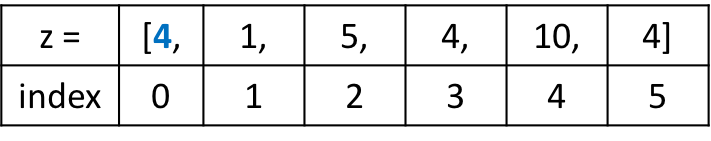
[7, 4, 3, 3]

* **clear()**: удаление всех элементов из списка
* **index(item)**: возвращает индекс элемента item. Если элемент не найден, генерирует исключение ValueError

>>> z = [4, 1, 5, 4, 10, 4]

>>> print(z.index(4))

0

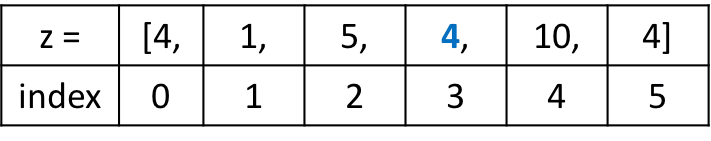


Вы также можете указать, откуда начинаете поиск.

КОПИРОВАТЬ

>>> print(z.index(4, 3))

3



* **pop([index])**: удаляет и возвращает элемент по индексу index. Если индекс не передан, то просто удаляет последний элемент.

### Метод Pop

z.pop(1) удаляет значение в индексе 1 и возвращает значение 4

Метод pop удаляет элемент в указанном индексе. Этот метод также вернет элемент, который был удален из списка. В случае, если вы не указали индекс, он по умолчанию удалит элемент по последнему индексу.

КОПИРОВАТЬ

z = [7, 4, 3, 3]

print(z.pop(1))

print(z)

4

[7, 3, 3]

* **count(item)**: возвращает количество вхождений элемента item в список

Метод count работает так, как звучит. Он считает количество раз, когда значение появляется в списке.

КОПИРОВАТЬ

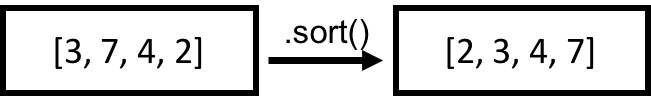
>>> random\_list = [4, 1, 5, 4, 10, 4]

>>> print(random\_list.count(4))

3

* **sort([key])**: сортирует элементы. По умолчанию сортирует по возрастанию. Но с помощью параметра key мы можем передать функцию сортировки.

### Метод Sort

Сортировка списка — фактическим кодом будем: z.sort()

Метод sort сортирует и меняет исходный список.

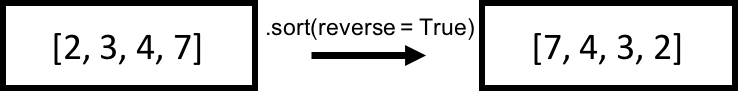
КОПИРОВАТЬ

z = [3, 7, 4, 2]

z.sort()

print(z)

[2, 3, 4, 7]

Сортировка списка с наибольшего значения к наименьшему

Вышеуказанный код сортирует список чисел от наименьшего к наибольшему. Код, указанный ниже, показывает, как вы можете сортировать список от наибольшего к наименьшему.

КОПИРОВАТЬ

# Сортировка и изменение исходного списка от наивысшего к наименьшему

z.sort(reverse = True)

print(z)

[7, 4, 3, 2]

Следует отметить, что вы также можете отсортировать [**список строк**](https://pythonru.com/osnovy/python-join) от А до Я (или A-Z) и наоборот.

КОПИРОВАТЬ

# Сортировка списка строками

names = ["Стив", "Рейчел", "Майкл", "Адам", "Джессика", "Лестер"]

names.sort()

print(names)

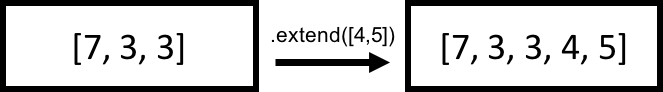
['Адам', 'Джессика', 'Лестер', 'Майкл', 'Рейчел', 'Стив']

* **reverse()**: расставляет все элементы в списке в обратном порядке

Кроме того, Python предоставляет ряд встроенных функций для работы со списками:

* **extend**

### Метод Extend



Метод extend расширяет список, добавляя элементы. Преимущество над append в том, что вы можете добавлять списки.  
Добавим [4, 5] в конец z:

КОПИРОВАТЬ

z = [7, 3, 3]

z.extend([4,5])

print(z)

[7, 3, 3, 4, 5]

То же самое можно было бы сделать, используя +.

КОПИРОВАТЬ

>>> print([1,2] + [3,4])

[7, 3, 3, 4, 5]

### Добавление и удаление элементов

Для добавления элемента применяются методы append() и insert, а для удаления - методы remove(), pop() и clear().

Использование методов:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | users = ["Tom", "Bob"]    # добавляем в конец списка  users.append("Alice")  # ["Tom", "Bob", "Alice"]  # добавляем на вторую позицию  users.insert(1, "Bill")          # ["Tom", "Bill", "Bob", "Alice"]    # получаем индекс элемента  i = users.index("Tom")  # удаляем по этому индексу  removed\_item = users.pop(i)            # ["Bill", "Bob", "Alice"]    last\_user = users[-1]  # удаляем последний элемент  users.remove(last\_user)           # ["Bill", "Bob"]    print(users)    # удаляем все элементы  users.clear() |

### Проверка наличия элемента

Если определенный элемент не найден, то методы remove и index генерируют исключение. Чтобы избежать подобной ситуации, перед операцией с элементом можно проверять его наличие с помощью ключевого слова **in**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | companies = ["Microsoft", "Google", "Oracle", "Apple"]  item = "Oracle"  # элемент для удаления  if item in companies:      companies.remove(item)    print(companies) |

Выражение item in companies возвращает True, если элемент item имеется в списке companies. Поэтому конструкция if item in companies может выполнить последующий блок инструкций в зависимости от наличия элемента в списке.

### Подсчет вхождений

Если необходимо узнать, сколько раз в списке присутствует тот или иной элемент, то можно применить метод count():

|  |  |
| --- | --- |
| 1  2  3  4 | users = ["Tom", "Bob", "Alice", "Tom", "Bill", "Tom"]    users\_count = users.count("Tom")  print(users\_count)      # 3 |

### Сортировка

Для сортировки по возрастанию применяется метод **sort()**:

|  |  |
| --- | --- |
| 1  2  3  4 | users = ["Tom", "Bob", "Alice", "Sam", "Bill"]    users.sort()  print(users)      # ["Alice", "Bill", "Bob", "Sam", "Tom"] |

Если необходимо отсортировать данные в обратном порядке, то мы можем после сортировки применить метод reverse():

|  |  |
| --- | --- |
| 1  2  3  4  5 | users = ["Tom", "Bob", "Alice", "Sam", "Bill"]    users.sort()  users.reverse()  print(users)      # ["Tom", "Sam", "Bob", "Bill", "Alice"] |

При сортировке фактически сравниваются два объекта, и который из них "меньше", ставится перед тем, который "больше". Понятия "больше" и "меньше" довольно условны. И если для чисел все просто - числа расставляются в порядке возрастания, то для строк и других объектов ситуация сложнее. В частности, строки оцениваются по первым символам. Если первые символы равны, оцениваются вторые символы и так далее. При чем цифровой символ считается "меньше", чем алфавитный заглавный символ, а заглавный символ считается меньше, чем строчный. Подробнее про сравнение строк описывалось в статье [Операции со строками](https://metanit.com/python/tutorial/2.5.php).

Таким образом, если в списке сочетаются строки с верхним и нижним регистром, то мы можем получить не совсем корректные результаты, так как для нас строка "bob" должна стоять до строки "Tom". И чтобы изменить стандартное поведение сортировки, мы можем передать в метод sort() в качестве параметра функцию:

|  |  |
| --- | --- |
| 1  2  3  4 | users = ["Tom", "bob", "alice", "Sam", "Bill"]    users.sort(key=str.lower)  print(users)      # ["alice", "Bill", "bob", "Sam", "Tom"] |

Кроме метода sort мы можем использовать встроенную функцию **sorted**, которая имеет две формы:

* sorted(list): сортирует список list
* sorted(list, key): сортирует список list, применяя к элементам функцию key

|  |  |
| --- | --- |
| 1  2  3  4 | users = ["Tom", "bob", "alice", "Sam", "Bill"]    sorted\_users = sorted(users, key=str.lower)  print(sorted\_users)      # ["alice", "Bill", "bob", "Sam", "Tom"] |

При использовании этой функции следует учитывать, что эта функция не изменяет сортируемый список, а все отсортированные элементы она помещает в новый список, который возвращается в качестве результата.

### Копирование списков

При копировании списков следует учитывать, что списки представляют изменяемый (mutable) тип, поэтому если обе переменных будут указывать на один и тот же список, то изменение одной переменной, затронет и другую переменную:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | users1 = ["Tom", "Bob", "Alice"]  users2 = users1  users2.append("Sam")  # users1 и users2 указывают на один и тот же список  print(users1)   # ["Tom", "Bob", "Alice", "Sam"]  print(users2)   # ["Tom", "Bob", "Alice", "Sam"] |

Это так называемое "поверхностное копирование" (shallow copy). И, как правило, такое поведение нежелательное. И чтобы происходило копирование элементов, но при этом переменные указывали на разные списки, необходимо выполнить глубокое копирование (deep copy). Для этого можно использовать метод **deepcopy()**, который определен во встроенном модуле **copy**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | import copy    users1 = ["Tom", "Bob", "Alice"]  users2 = copy.deepcopy(users1)  users2.append("Sam")  # пееменные users1 и users2 указывают на разные списки  print(users1)   # ["Tom", "Bob", "Alice"]  print(users2)   # ["Tom", "Bob", "Alice", "Sam"] |

### Копирование части списка

Если необходимо скопировать не весь список, а только его какую-то определенную часть, то мы можем применять специальный синтаксис. который может принимать следующие формы:

* list[:end]: через параметр end передается индекс элемента, до которого нужно копировать список
* list[start:end]: параметр start указывает на индекс элемента, начиная с которого надо скопировать элементы
* list[start:end:step]: параметр step указывает на шаг, через который будут копироваться элементы из списка. По умолчанию этот параметр равен 1.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | users = ["Tom", "Bob", "Alice", "Sam", "Tim", "Bill"]    slice\_users1 = users[:3]   # с 0 по 3  print(slice\_users1)   # ["Tom", "Bob", "Alice"]    slice\_users2 = users[1:3]   # с 1 по 3  print(slice\_users2)   # ["Bob", "Alice"]    slice\_users3 = users[1:6:2]   # с 1 по 6 с шагом 2  print(slice\_users3)   # ["Bob", "Sam", "Bill"] |

| **Метод** | **Описаниее** |
| --- | --- |
| x in s | True если элемент x находится в списке s |
| x not in s | True если элемент x не находится в списке s |
| s1 + s2 | Объединение списков s1 и s2 |
| s \* n , n \* s | Копирует список s n раз |
| len(s) | Длина списка s, т.e. количество элементов в s |
| min(s) | Наименьший элемент списка s |
| max(s) | Наибольший элемент списка s |
| sum(s) | Сумма чисел списка s |
| for i in list() | Перебирает элементы слева направо в [**цикле for**](https://pythonru.com/osnovy/sintaksis-vozmozhnosti-i-podvodnye-kamni-cikla-for-v-python-3) |

Примеры использование функций со списками:

КОПИРОВАТЬ

>>> list1 = [2, 3, 4, 1, 32]

>>> 2 in list1 # 2 в list1?

True

>>> 33 not in list1 # 33 не в list1?

True

>>> len(list1) # количество элементов списка

5

>>> max(list1) # самый большой элемент списка

32

>>> min(list1) # наименьший элемент списка

1

>>> sum(list1) # сумма чисел в списке

42

# генератор списков python (list comprehension)

>>> x = [i for i in range(10)]

>>> print(x)

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> print(list1.reverse()) # разворачивает список

[32, 1, 4, 3, 2]

### Операторы + и \* для списков

+ объединяет два списка.

КОПИРОВАТЬ

list1 = [11, 33]

list2 = [1, 9]

list3 = list1 + list2

print(list3)

[11, 33, 1, 9]

\* копирует элементы в списке.

КОПИРОВАТЬ

list4 = [1, 2, 3, 4]

list5 = list4 \* 3

print(list5)

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

### Оператор in и not in

Оператор in проверяет находится ли элемент в списке. При успешном результате он возвращает True, в случае неудачи, возвращает False .

КОПИРОВАТЬ

>>> list1 = [11, 22, 44, 16, 77, 98]

>>> 22 in list1

True

Аналогично not in возвращает противоположный от оператора in результат.

КОПИРОВАТЬ

>>> 22 not in list1

False