Список (list) в Python

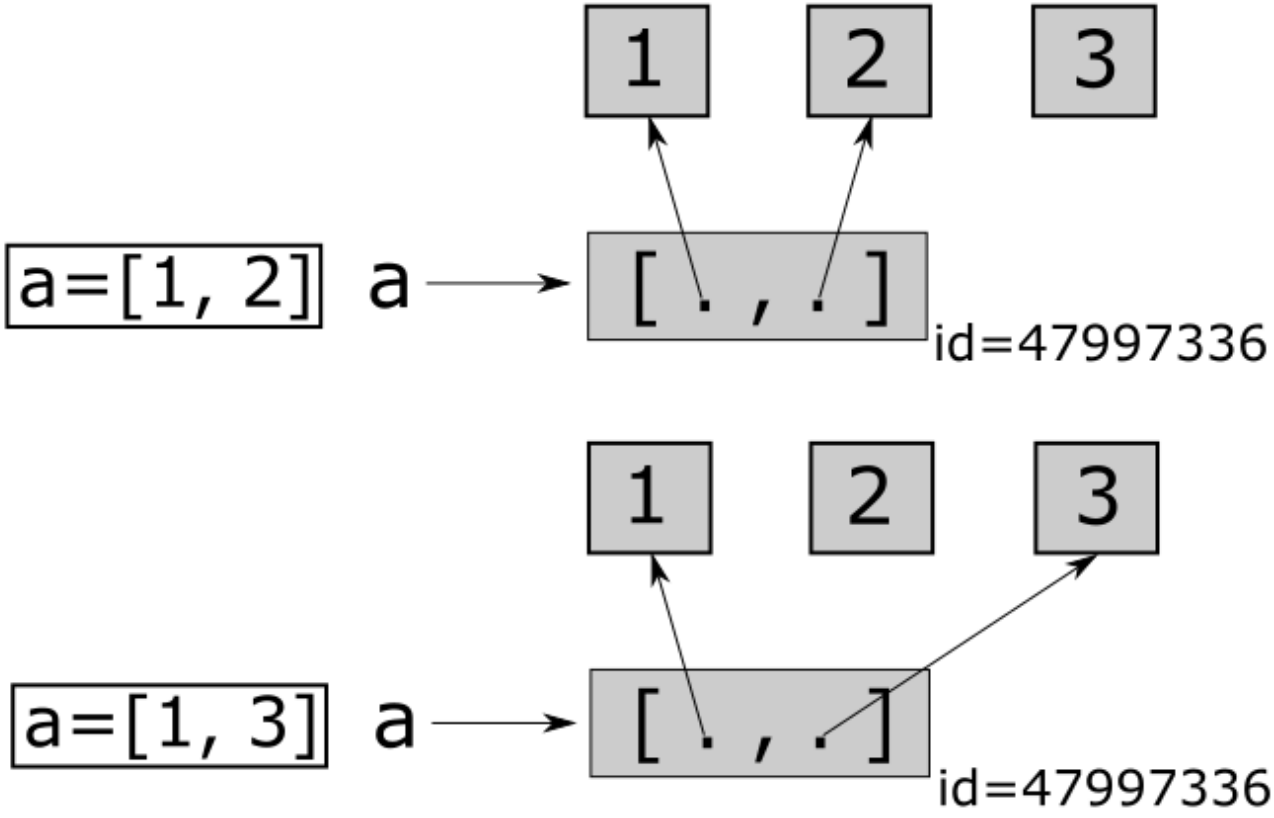
Одна из ключевых особенностей Python, благодаря которой он является таким популярным – это простота. Особенно подкупает простота работы с различными структурами данных – списками, кортежами, словарями и множествами. Сегодня мы рассмотрим работу со списками.

Что такое список (list) в Python?

Список (`list`) – это структура данных для хранения объектов различных типов. Если вы использовали другие языки программирования, то вам должно быть знакомо понятие массива. Так вот, список очень похож на массив, только, как было уже сказано выше, в нем можно хранить объекты различных типов. Размер списка не статичен, его можно изменять. Список по своей природе является изменяемым типом данных. Про типы данных можно подробно прочитать здесь. Переменная, определяемая как список, содержит ссылку на структуру в памяти, которая в свою очередь хранит ссылки на какие-либо другие объекты или структуры.

Как списки хранятся в памяти?

Как уже было сказано выше, список является изменяемым типом данных. При его создании в памяти резервируется область, которую можно условно назвать некоторым “контейнером”, в котором хранятся ссылки на другие элементы данных в памяти. В отличии от таких типов данных как число или строка, содержимое “контейнера” списка можно менять. Для того, чтобы лучше визуально представлять себе этот процесс взгляните на картинку ниже. Изначально был создан список содержащий ссылки на объекты 1 и 2, после операции a[1] = 3, вторая ссылка в списке стала указывать на объект 3.



Более подробно эти вопросы обсуждались в уроке 3 (Типы и модель данных).

Создание, изменение, удаление списков и работа с его элементами

Создать список можно одним из следующих способов.

```
a = []
print(type(a))
# >>> <class 'list'>

b = list()
print(type(b))
# >>> <class 'list'>
```

Также можно создать список с заранее заданным набором данных.

```
a = [1, 2, 3]
print(type(a))
# >>> <class 'list'>
```

Если у вас уже есть список и вы хотите создать его копию, то можно воспользоваться следующим способом:

```
a = [1, 3, 5, 7]
b = a[:]
print(a)
# >>> [1, 3, 5, 7]
print(b)
# >>> [1, 3, 5, 7]
```

или сделать это так:

```
a = [1, 3, 5, 7]
b = list(a)
print(a)
# >>> [1, 3, 5, 7]
print(b)
# >>> [1, 3, 5, 7]
```

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, а не копия списка a. Т.е. если вы будете изменять список `a`, то и `b` тоже будет меняться.

```
a = [1, 3, 5, 7]
b = a
print(a)
# >>> [1, 3, 5, 7]
print(b)
# >>> [1, 3, 5, 7]
a[1] = 10
print(a)
# >>> [1, 10, 5, 7]
print(b)
# >>> [1, 10, 5, 7]
```

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

Для удаления элемента из списка, в случае, если вы знаете его значение, используйте метод `remove(x)`, при этом будет удалена первая ссылка на данный элемент.

Если необходимо удалить элемент по его индексу, воспользуйтесь командой `del имя_списка[индекс]`.

Изменить значение элемента списка, зная его индекс, можно напрямую к нему обратившись.

```
d = [2, 4, 9]
print(d)
# >>> [2, 4, 9]

d[1] = 17
 print(d)
# >>> [2, 17, 9]
```

Очистить список можно просто заново его проинициализировав, так как будто вы его вновь создаете. Для получения доступа к элементу списка укажите индекс этого элемента в квадратных скобках.

```
a = [3, 5, 7, 10, 3, 2, 6, 0]
a[2]
# >>> 7
```

Можно использовать отрицательные индексы, в таком случае счет будет идти с конца, например для доступа к последнему элементу списка можно использовать вот такую команду:

```
a[-1]
# >>> 0
```

Для получения из списка некоторого подсписка в определенном диапазоне индексов, укажите начальный и конечный индекс в квадратных скобках, разделив их двоеточием.

```
a[1:4]
# >>> [5, 7, 10]
```

Методы списков

list.append(x)

Добавляет элемент в конец списка. Ту же операцию можно сделать так а `[len(a):] = [x]`.

```
a = [1, 2]
a.append(3)
print(a)
# >>> [1, 2, 3]
```

list.extend(L)

Расширяет существующий список за счет добавления всех элементов из списка `L`. Эквивалентно команде а `[len(a):] = L`.

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

list.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]
```

list.remove(x)

Удаляет первое вхождение элемента `x` из списка.

```
a = [1, 2, 3]
a.remove(1)
```

```
print(a)
# >>> [2, 3]
```

del x[i]

Если необходимо удалить элемент по его индексу, воспользуйтесь командой `del имя_списка[индекс]`.

```
c = [3, 5, 1, 9, 6]
print(c)
# >>> [3, 5, 1, 9, 6]
del c[2]
print(c)
# >>> [3, 5, 9, 6]
```

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]
```

list.clear()

Удаляет все элементы из списка. Эквивалентно `del a[:]`.

```
a = [1, 2, 3, 4, 5]
print(a)
# >>> [1, 2, 3, 4, 5]
a.clear()
print(a)
# >>> []
```

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

Возвращает индекс элемента.

```
a = [1, 2, 3, 4, 5]
a.index(4)
# >>> 3
```

list.count(x)

Возвращает количество вхождений элемента x в список.

```
a=[1, 2, 2, 3, 3]
print(a.count(2))
# >>> 2
```

list.sort(key=None, reverse=False)

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]
```

list.reverse()

Изменяет порядок расположения элементов в списке на обратный.

```
a = [1, 3, 5, 7]
a.reverse()
print(a)
# >>> [7, 5, 3, 1]
```

list.copy()

Возвращает копию списка. Эквивалентно: `a[:]`.

```python
a = [1, 7, 9]
b = a.copy()
print(a)
# >>> [1, 7, 9]
print(b)
# >>> [1, 7, 9]
b[0] = 8
print(a)
# >>> [1, 7, 9] # Осталось прежним
print(b)
# >>> [8, 7, 9] # Изменилось
```

List Comprehensions

List Comprehensions чаще всего на русский язык переводят как абстракция списков или списковое включение, является частью синтаксиса языка, которая предоставляет простой способ построения списков. Тема обязательна к изучению.

Слайсы / Срезы

Слайсы (срезы) являются очень мощной составляющей Python, которая позволяет быстро и лаконично решать задачи выборки элементов из списка. Выше уже был пример использования слайсов, здесь разберем более подробно работу с ними. Создадим список для экспериментов:

```python
a = [i for i in range(10)]
```

Слайс задается тройкой чисел, разделенных запятой: `start:stop:step`.

- `Start` - позиция с которой нужно начать выборку
- `stop` - конечная позиция
- `step` - шаг.

При этом необходимо помнить, что выборка не включает элемент определяемый stop.

Рассмотрим примеры:

```python
# 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]
```

Слайсы можно сконструировать заранее, а потом уже использовать по мере необходимости. Это возможно сделать, в виду того, что слайс - это объект класса `slice`. Ниже приведен пример, демонстрирующий эту функциональность:

```
s = slice(0, 5, 1)
a[s]
# >>> [0, 1, 2, 3, 4]

s = slice(1, 8, 2)
a[s]
# >>> [1, 3, 5, 7]
```

источник

Теги: python types list

Отредактировать эту страницу

*Последнее обновление **4 сент. 2023 г.** от **Stavis***