

Лекция 3. Коллекции: списки и кортежи

Списки и операции над ними. Срезы списков

Тип range и создание последовательностей

Кортежи и их отличие от списков

Распаковка списков и кортежей в переменные





Списки

Списки и операции над ними. Срезы списков. Тип range и создание последовательностей

Списки

Это упорядоченная последовательность элементов.

- ❑ Любой элемент в списке можно получить по его порядковому номеру
- ❑ Нумерация в списке начинается с 0
- ❑ Элементы в списке разделяются запятыми



Списки (продолжение)

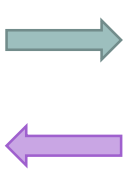
Строки – это тоже списки:

`s = 'Мама мыла раму' →`

`l = ['М', 'а', 'м', 'а', ' ', 'м', 'ы', 'л', 'а', ' ', 'р', 'а', 'м', 'у']`

`l[10] = l[-4] = 'р'`

Перемещаться по списку можно из начала в конец или из конца в начало:



М	а	м	а		м	ы	л	а		р	а	м	у
0	1	2	3	4	5	6	7	8	9	10	11	12	13
-14	-13	-12	-11	-10	-9	-7	-7	-6	-5	-4	-3	-2	-1



Создание списка

Создание пустого списка

`my_list = list() → []`

`my_list = [] → []`

Создание непустого списка

вручную:

`my_list = [1, 2, 3] → [1,2,3]`

из другого типа данных:

`my_list = list("Мама") → ["М","а","м","а"]`



Многомерные списки

Внутри списка могут быть другие списки.

Например, список дел:

```
todo_list = [  
0 ["Посмотреть лекцию по условиям и циклам", 1, "час"],  
1 ["Сделать ДЗ Python", 30, "мин"]  
]  
      0           1     2
```

`todo_list[1] → ["Сделать ДЗ Python", 30, "мин"]`

`todo_list[1][1] → 30`



Операции со списками

len(list)	Количество элементов в списке (длина списка)	<code>my_list = [1, 2]</code> <code>len(my_list) → 2</code>
list.append()	Добавление элемента в список элемент всегда добавляется в конец	<code>my_list = [1, 2]</code> <code>my_list.append(3) → [1, 2, 3]</code>
list.insert(i, X)	Вставить элемент X в любое место списка Внимание! номер элемента i должен быть от 0 до (длина списка – 1)	<code>my_list = [1, 2]</code> <code>my_list.insert(1, 3) → [1, 3, 2]</code>
list.extend(L2) или просто L1 + L2	Соединяет два списка второй список L2 вставляется в конец первого списка. Также можно просто сложить два списка L1 и L2	<code>my_list = [1, 2]</code> <code>my_list2 = [3]</code> <code>my_list.extend(my_list2) → [1, 2, 3]</code> или <code>my_list.extend([3]) → [1, 2, 3]</code> <code>my_list + my_list2 → [1, 2, 3]</code>



Операции со списками (продолжение)

list.remove(X)	Удаляет первый элемент в списке с значением X если таких значений несколько, удалится первое. Если их вообще нет, код упадет с ошибкой ValueError («ошибка значения»)	<code>my_list = [1, 2]</code> <code>my_list.remove(2) → [1]</code> <code>my_list.remove(3) → ValueError!</code>
list.pop(i)	«Вынимает» элемент под номером i отличие от remove() в том, что этот элемент можно куда-то сохранить	<code>my_list = [1, 2]</code> <code>save = my_list.pop(0) → [2]</code> при этом 1 запишется в save
list.count(X)	Подсчитывает количество элементов с значением X	<code>my_list = [1, 2, 1]</code> <code>my_list.count(1) → 2</code>
list.clear()	Очищает список эта операция ничего не возвращает	<code>my_list = [1, 2]</code> <code>my_list.clear() → в my_list будет []</code>



Операции со списками (продолжение)

list.index(X) list.index(X, start, end)	Возвращает индекс элемента X, если такой есть в списке start — искать не с начала, end — искать не до конца. Если такого элемента нет, код упадет с ошибкой ValueError	<code>my_list = [1, 2, 1]</code> <code>my_list.index(1)</code> → 0 (число 1 первый раз находится по индексу 0) <code>my_list.index(1,1)</code> → 2 (мы пропустили первую 1, зато нашли следующую) <code>my_list.index(1,1,2)</code> → ValueError (мы задали поиск с индекса 1 до 2, но в этом промежутке есть только число 2)
list.reverse()	Разворачивает список в обратном направлении эта операция ничего не возвращает	<code>my_list = [1, 2]</code> <code>my_list.reverse()</code> → в <code>my_list</code> будет <code>[2, 1]</code> . помните, что она ничего не возвращает
reversed(list)	Возвращает список, развернутый в обратном направлении возвращает развернутый список-итератор. Исходный список остается без изменений	<code>my_list = [1, 2]</code> <code>reversed(my_list)</code> → вернет тип <list_reverseiterator> , который нужно преобразовать в list и получится <code>[2, 1]</code>



Операции со списками (продолжение)

list.sort() list.sort(reverse=True)	Сортирует список по умолчанию сортирует по возрастанию. Если добавить reverse=True , сортирует по убыванию	<code>my_list = [1, 2, 0.4]</code> <code>my_list.sort()</code> → в <code>my_list</code> будет <code>[0.4, 1, 2]</code> <code>my_list.sort(reverse=True)</code> → в <code>my_list</code> будет <code>[2, 1, 0.4]</code> будьте внимательны со списком строк! <code>my_list = ["раз", "два", "три"]</code> <code>my_list.sort()</code> → в <code>my_list</code> будет <code>['два', 'раз', 'три']</code> , потому что номера букв "д" < "р" < "т"
list.copy()	Создает копию списка у нас появляется второй такой же список, никак не связанный с первым списком. Зачем? Чтобы случайно не испортить первый список при вычислениях	<code>my_list = [1, 2]</code> <code>my_list2 = my_list</code> <code>my_list2[1] = 3</code> → в <code>my_list</code> будет <code>[1, 3]</code> а вот так правильно: <code>my_list = [1, 2]</code> <code>my_list2 = my_list.copy()</code> <code>my_list2[1] = 3</code> → в <code>my_list2</code> будет <code>[1, 3]</code> , в <code>my_list</code> <code>[1, 2]</code>



Взаимодействие списков с другими типами

Списки и строки хорошо работают друг с другом:

<code>list(string)</code>	Можно преобразовать строку в список	<code>s = "Мама"</code> <code>list(s) → ["М", "а", "м", "а"]</code>
<code>"разделитель".join(list)</code>	Можно преобразовать список обратно в строку если нужно просто склеить, вместо разделителя пишем "" (пустые кавычки) или ' ' (пустые апострофы). Если нужен разделитель, указываем его, например, "+"	<code>my_list = ["М", "а", "м", "а"]</code> <code>"".join(my_list) → "Мама"</code> <code>"+".join(my_list) → "М+а+м+а"</code>
<code>list * N</code>	Дублирование («умножение») списка список можно «умножить» на число. В результате получится список со списками	<code>la = ["Ла"]</code> <code>la * 3 → [["Ла"],["Ла"],["Ла"]]</code>



Срезы списков

Это часть, которую мы «вырезаем» из любого места списка

list[start:stop:step]
list[start:]
list[:stop]

Полностью настраиваемый срез

start – индекс первого элемента среза,

stop – индекс элемента, перед которым срез закончится,

step – шаг среза (step = 1 – берем каждый элемент, step = 2 – берем каждый второй элемент и т.д. По умолчанию **step = 1**).

Любой из параметров можно опустить, но **двоеточие** опускать нельзя!

Примеры:

[:] или **::]** или **::1]** – весь список

::2] – каждый второй элемент списка (четные индексы: 0, 2, 4 и т.д.)

1::2] – каждый второй элемент списка (нечетные индексы: 1, 3, 5 и т.д.)

[5:] – часть списка от элемента с индексом 5 (6й по счету, т.к. индексы списка начинаются с 0) и до конца

[:5] – список с начала до элемента с индексом 4 включительно (5 элементов)

[2:5] – список от 2го до 4го индекса (3 элемента)



Срезы списков (продолжение)

<code>[:]</code> или <code>::</code> или <code>::1</code> – весь список	<pre>my_list = [1, 2, 3, 4, 5, 6, 7, 8] my_list[:] → [1, 2, 3, 4, 5, 6, 7, 8]</pre>
<code>::2</code> – каждый второй элемент списка (нечетные индексы) <code>[1::2]</code> – каждый второй элемент списка (четные индексы)	<pre>my_list[::2] → [1, 3, 5, 7] my_list[1::2] → [2, 4, 6, 8]</pre>
<code>[5:]</code> – часть списка от элемента с индексом 5 (6й по счету, т.к. индексы списка начинаются с 0) и до конца	<pre>my_list[5:] → [6, 7, 8]</pre>
<code>[:5]</code> – список с начала до элемента с индексом 4 включительно (будет 5 элементов)	<pre>my_list[:5] → [1, 2, 3, 4, 5]</pre>
<code>[2:5]</code> – список от 2го до 4го индекса (будет 3 элемента)	<pre>my_list[2:5] → [3, 4, 5]</pre>



Срезы списков (продолжение)

Срезы списка могут быть и отрицательными

list[start:stop:step]	<p>Полностью настраиваемый срез start – индекс первого элемента среза, stop – индекс элемента, перед которым срез закончится, step – шаг среза. step может быть отрицательным (тогда шаг идет в обратном порядке). Внимание! Если указывать срез от большего к меньшему (от 5 до 1 или от -1 до -4), то step должен быть отрицательным и step указывать обязательно!</p>	<p>Примеры с отрицательными индексами: [::-1] – весь список в обратном порядке (step = -1) [-4:-1] – элементы списка от 4го с конца до предпоследнего (3 элемента) [-1:-4:-1] – элементы от последнего до 3го с конца в обратном порядке (3 элемента)</p>
------------------------------	--	---



Срезы списков (продолжение)

Примеры с отрицательными индексами

Исходный список	<code>my_list = [1, 2, 3, 4, 5, 6, 7, 8]</code>
<code>[::-1]</code> – весь список в обратном порядке (step = -1)	<code>my_list[::-1] → [8, 7, 6, 5, 4, 3, 2, 1]</code>
<code>[-4:-1]</code> – элементы списка от 4го с конца до предпоследнего (3 элемента)	<code>my_list[-4:-1] → [5, 6, 7]</code>
<code>[-1:-4:-1]</code> – элементы от последнего до 3го с конца в обратном порядке (3 элемента)	<code>my_list[-1:-4:-1] → [8, 7, 6]</code>



Срезы списков (продолжение)

`my_list[:3]`
или `my_list[:-5]`

`my_list[4:6]`
или `my_list[-4:-2]`

`my_list[6:]`
или `my_list[-2:]`

1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7
-8	-7	-6	-5	-4	-3	-2	-1





Тип `range` и создание последовательностей



Тип `range`

`range` — это специальный тип, который создается функцией `range()` и представляет собой **последовательность**.

- ❑ `range` нужен, когда нужно быстро создать последовательность чисел любой длины
- ❑ Числа в `range` — только целые
- ❑ Числа в `range` могут быть и положительными, и отрицательными
- ❑ `range` — это арифметическая прогрессия (каждый следующий элемент равен предыдущему + какое-то целое число)
- ❑ `range` можно превратить в список
- ❑ `range` обычно используется в циклах



Создание последовательностей с `range`

<code>range(start:stop:step)</code>	Создание последовательности start – значение первого элемента, stop – значение, перед которым последовательность закончится, step – шаг приращения. По умолчанию step = 1 . step может быть отрицательным	<code>range(10)</code> – создаст последовательность чисел 0..9 <code>range(1, 10)</code> – создаст последовательность 1..9 <code>range(0, 10, 2)</code> – создаст последовательность четных чисел 0..8 <code>range(1, 10, 2)</code> – создаст последовательность нечетных чисел 1..9 <code>range(-1, -10, -1)</code> – создаст последовательность отрицательных чисел -1..-9
<code>list(r)</code>	Преобразует <code>range</code> в <code>list</code>	<code>r = range(5) → range(0, 5)</code> <code>list(r) → [0, 1, 2, 3, 4]</code>





Кортежи

Кортежи и их отличие от списков

Кортеж

Кортеж — это список, который **нельзя изменить**

- ❑ Кортежи используются для защиты данных от случайного изменения (нельзя удалять, добавлять или перезаписывать значения кортежа)
- ❑ Кортеж занимает меньше памяти, чем такой же список
- ❑ Кортеж работает быстрее, чем список
- ❑ Кортеж можно использовать в качестве ключа словаря*, а список — нельзя
- ❑ Во всем остальном кортежи работают по тем же правилам, что и списки. Все функции списков, которые не изменяют данные, работают и для кортежей



Правила для кортежей

Кортеж — это список, который **нельзя изменить**

- ❑ Тип «кортеж» в Python называется tuple
- ❑ Кортеж выводится в круглых скобках: ()
- ❑ Кортеж создается один раз. Изменить его нельзя. Если попытаетесь — получите ошибку
- ❑ Элементы в кортеже начинаются с 0
- ❑ Элементы в кортеже записываются через запятую



Создание кортежа

Создание пустого кортежа

`my_tuple = tuple() → ()`

`my_tuple = () → ()`

Создание непустого кортежа

вручную:

`my_tuple = (1, 2, 3) → (1,2,3)`

из другого типа данных:

`my_tuple = tuple("Мама") → ("М","а","м","а")`

`my_tuple = tuple([1, 2, 3]) → (1, 2, 3)`





Дополнительные функции

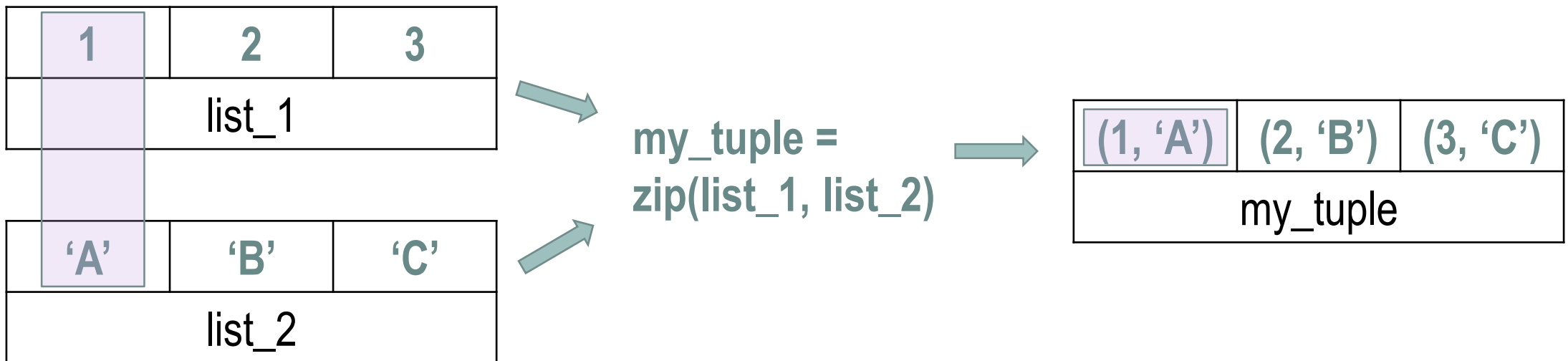
Функция zip

Проверка вхождения в список/кортеж

Распаковка списка/кортежа

Функция zip

Функция `zip(list_1, list_2, ...)` берет на вход несколько списков или кортежей и делает составной объект, состоящий из кортежей. Первый кортеж содержит все первые элементы входных списков, второй – вторые и так далее



Операторы проверки вхождения

in возвращает True, если элемент входит в список/кортеж

not in возвращает True, если элемент НЕ входит в список/кортеж

```
names = ["Мария", "Петр", "Александр", "Лилия"]  
print("Мария" in names)  
print("Елена" not in names)
```

```
True  
True
```



Распаковка списков и кортежей

Элементы списка/кортежа можно «разобрать» на переменные без использования циклов, обычным присваиванием.

Пишем столько переменных, сколько нам нужно. «Лишнее» остается в списке или кортеже – эту переменную нужно пометить звездочкой¹

```
my_list = [1, 2, 3, 4, 5]  
v1, v2, *v3 = my_list  
print(v1, v2, v3)
```



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

¹ - * и ** мы подробно разберем в лекции про функции



Распаковка списков и кортежей (продолжение)

Результат сцепления функцией `zip` можно использовать в циклах

```
names = ["Мария", "Петр", "Александр", "Лилия"]  
ages = [25, 17, 38, 33]  
▼ for n, a in zip(names, ages):  
    print(f"Возраст {n}: {a} лет")
```

```
Возраст Мария: 25 лет  
Возраст Петр: 17 лет  
Возраст Александр: 38 лет  
Возраст Лилия: 33 лет
```

