

Коллекции данных: множества

Александр Бардин



Александр Бардин

Начальник отдела Python-разработки в
Open Solutions



Вспоминаем прошрое занятие

Вопрос: как найти все общие элементы
в двух списках?



Вспоминаем прошрое занятие

Вопрос: как найти все общие элементы
в двух списках?

Ответ:

```
for l1 in list1:
```

```
    for l2 in list2:
```

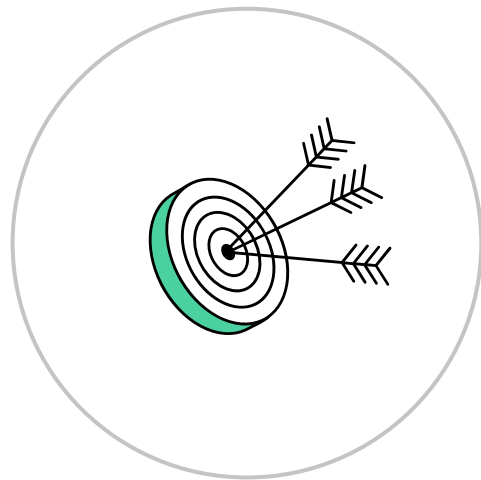
```
        if l1 == l2:
```

```
            print("Общий")
```



Цели занятия

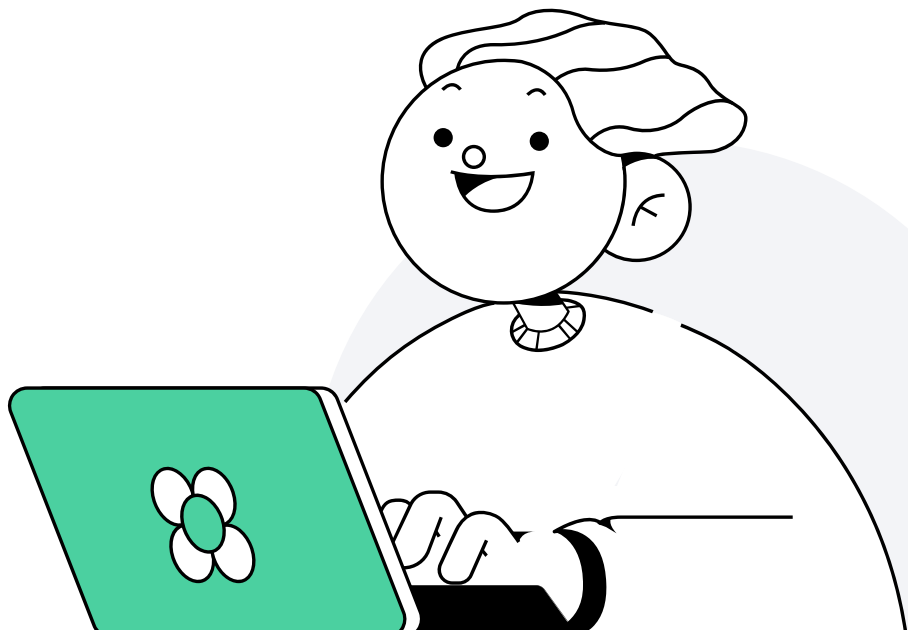
- Изучим, что такое множество и как оно устроено
- Сравним списки и множества
- Вспомним теорию множеств
- Разберём, как работать с множествами в Python
- Научимся быстро сопоставлять большие выборки данных



План занятия

- 1 Множество
- 2 Теория множеств
- 3 Операции над множествами в Python
- 4 Итоги
- 5 Домашнее задание

*Нажми на нужный раздел для перехода



Множество

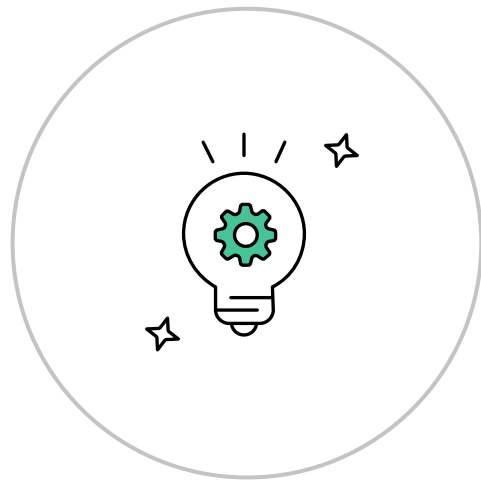
Что такое множество и как оно устроено



1



Множество — неупорядоченная совокупность произвольных **уникальных элементов**



Множество и список

Список

[1, 2, 3]

Множество

{1, 2, 3}

Список

[1, 2, 3, 1]

Множество

{1, 2, 3}

Список

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

Множество

Нельзя

Создание множества

Создать пустое множество:

```
my_set = set() → {}
```

Создать множество с элементами:

```
my_set = {1, 2, 3} → {1, 2, 3}
```

Создать множество из списка:

```
my_list = [1, 2, 3, 1]
```

```
my_set = set(my_list) → {1, 2, 3}
```

Правила для множеств

- Множество создаётся функцией **set()**
- Множество записывается в фигурных скобках {}
- Элементы разделяются запятыми
- Порядок элементов в множестве не соблюдается.
Получить значение по индексу (как в списке) — нельзя
- Множество обычно преобразовывают в список
- Элементы в множестве уникальны (не могут повторяться).
Если преобразовать список в множество, повторяющиеся значения исчезнут
- Элементы множества реализуют теорию множеств

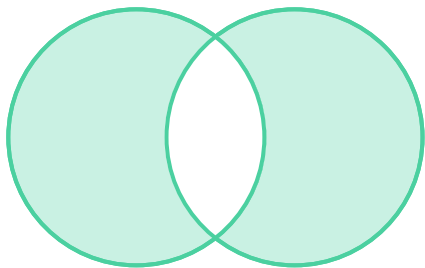
Теория множеств

Операции теории множеств и визуализация операций
над множествами

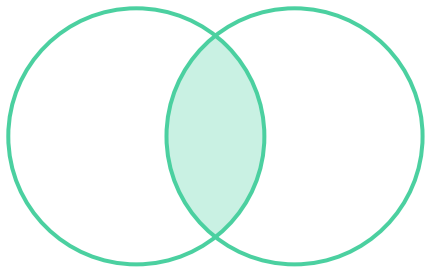


2

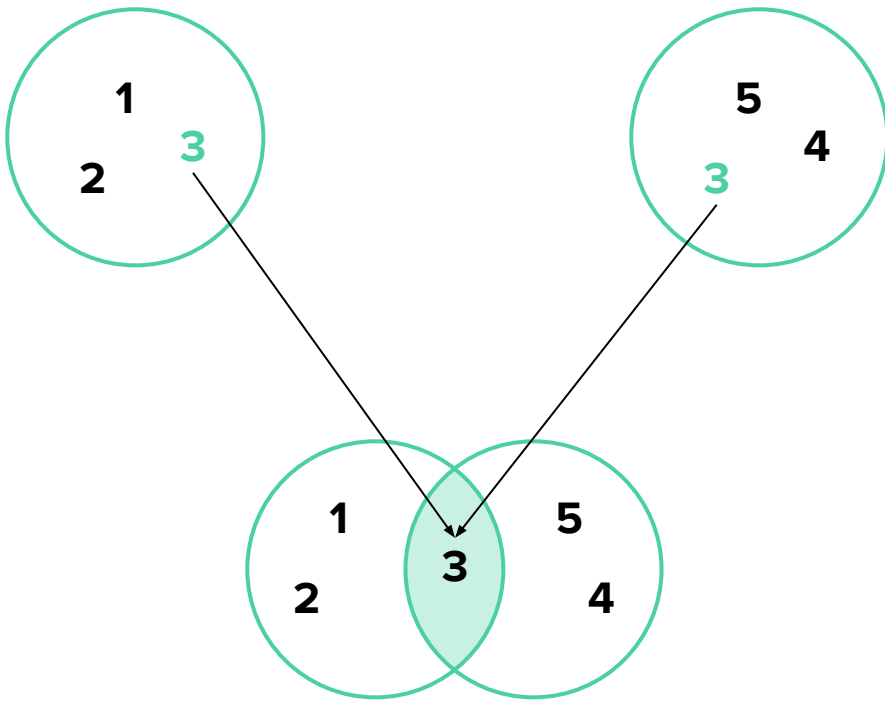
Круги Эйлера — диаграммы Венна



Несовпадающие элементы

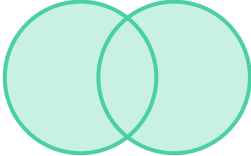
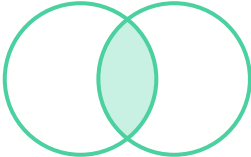
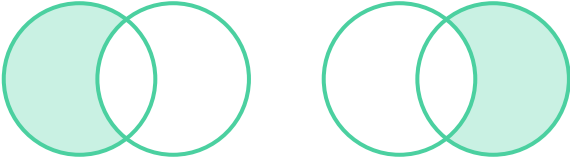


Совпадающие элементы



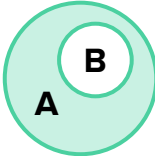
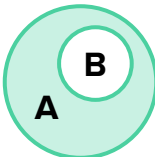
Теория множеств

Операции: объединение, пересечение, разность

Операция	Название	Визуализация	Пример
\cup	Объединение		$A = (1, 2, 3)$ $B = (3, 4, 5)$ $A \cup B \rightarrow (1, 2, 3, 4, 5)$
\cap	Пересечение		$A = (1, 2, 3)$ $B = (3, 4, 5)$ $A \cap B \rightarrow (3)$
$-$	Разность — важен порядок аргументов		$A = (1, 2, 3)$ $B = (3, 4, 5)$ $A - B \rightarrow (1, 2)$ $B - A \rightarrow (4, 5)$

Теория множеств

Операции: симметрическая разность, проверка вхождения

Операция	Название	Визуализация	Пример
Δ	Симметрическая разность		$A = (1, 2, 3)$ $B = (3, 4, 5)$ $A \Delta B \rightarrow (1, 2, 4, 5)$
\subset	Подмножество		$A = (1, 2, 3, 4, 5)$ $B = (3, 4)$ $A \subset B \rightarrow \text{Нет}$ $B \subset A \rightarrow \text{Да}$
\supset	Аналогичного названия в русском языке нет. По смыслу это «родительское множество»		$A = (1, 2, 3, 4, 5)$ $B = (3, 4)$ $A \supset B \rightarrow \text{Да}$ $B \supset A \rightarrow \text{Нет}$

Операции над множествами в Python

Как реализована теория множеств в Python



3

Виды операций над множествами

Общие операции:

`len()`, `copy()`, `clear()`

Операции над элементами множеств:

`in`, `add()`, `remove()`, `discard()`

Сравнение множеств:

`==`, `issubset()`, `issuperset()`

Операции теории множеств:

`union()`, `intersection()`, `difference()`, `symmetric_difference()`, `update()`

Общие операции

len()

Подсчитывает количество элементов в множестве

```
a_set = {1, 2, 3}
len(a_set)
```

```
ooo
```

```
3
```

copy()

Создаёт копию данных словаря (так же, как для списков и множеств)

```
a_set = {1, 2, 3}
b_set = a_set.copy()
id(a_set)
id(b_set)
```

```
ooo
```

```
140561113241856
140561113242080
```

clear()

Очищает содержимое множества

```
a_set = {1, 2, 3}
a_set.clear()
```

```
ooo
```

```
set()
```

Операции над элементами множеств

X in set

Проверяет наличие элемента в множестве

```
a_set = {1, 2, 3}
1 in a_set
4 in a_set
```

ooo

```
True
False
```

add(X)

Добавляет элемент X в множество, если этого элемента в множестве ещё нет

```
a_set = {1, 2, 3}
a_set.add(4)
```

ooo

```
{1, 2, 3, 4}
```

remove(X)

Удаляет элемент X из множества. Обратите внимание: если этого элемента в множестве нет, код упадёт с ошибкой KeyError

```
a_set = {1, 2, 3}
a_set.remove(2)
a_set.remove(4)
```

ooo

```
{1, 3}
```

```
KeyError
```

discard(X)

Удаляет элемент X из множества. Обратите внимание: если этого элемента в множестве нет, код не упадёт с ошибкой KeyError

```
a_set = {1, 2, 3}
a_set.discard(4)
```

ooo

```
{1, 2, 3}
```

Сравнение множеств

`set1 == set2`

Сравнение двух множеств. Если множества полностью совпадают по составу элементов, возвращает True, иначе — False

```
a_set = {1, 2}
b_set = {2, 1}
c_set = {1, 3}
a_set == b_set
a_set == c_set
```

ooo

True

False

`set1.issubset(set2)`

Проверяет, является ли set1 подмножеством set2. set1 должно входить в set2, либо они должны быть идентичны

```
a_set = {1, 2, 3, 4}
b_set = {1, 2}
b_set.issubset(a_set)
a_set.issubset(b_set)
```

ooo

True

False

`set1.issuperset(set2)`

Проверяет, включает ли в себя множество set1 множество set2. set2 должно входить в set1, либо они должны быть идентичны

```
a_set = {1, 2, 3, 4}
b_set = {1, 2}
b_set.issuperset(a_set)
a_set.issuperset(b_set)
```

ooo

False

True

Операции теории множеств

`set1.union(set2, ...)` или `set1 | set2 | ...`

Объединяет `set1` и любое количество множеств в скобках.

В результате создаётся новое множество

```
set1 = {1, 2}
set1.union({4, 5}, {3})
set1 | {4, 5} | {3}
```

```
ooo
{1,2,3,4,5}
{1,2,3,4,5}
```

`set1.intersection(set2, ...)` или `set1 & set2 & ...`

Пересекает `set1` с любым количеством множеств. В результате создаётся новое множество, содержащее элементы, которые совпали во всех без исключения множествах. Если таких совпадений нет, создаётся пустое множество `{}`

```
set1 = {1, 2}
set1.intersection({1, 5}, {3})
set1.intersection({1, 5}, {1, 3})
set1 & {1, 5} & {1, 3}
```

```
ooo
{}
{1}
```

`set1.difference(set2, ...)` или `set1 - set2 - ...`

Вычитает из `set1` все элементы, присутствующие в множествах `set2` и др.

Создаёт новое множество

```
set1 = {1, 2, 3, 4, 5}
set1.difference({1, 5}, {3})
{6, 1} - set1
```

```
ooo
{2, 4}
{6}
```

`set1.symmetric_difference(set2)` или `set1 ^ set2`

Симметрическая разность множеств оставляет в `set1` и `set2` элементы, которые встречаются только в одном из них.

Создаёт новое множество

```
set1 = {1, 2}
set1.symmetric_difference({3, 2})
set1 ^ {3, 2}
```

```
ooo
{1,3}
{1, 3}
```

Операции теории множеств с update и сокращённая запись

Чтобы не создавать новое множество, а записать результат в существующее, используйте функции вида «update». Либо короткую запись вида «|=»

set1.update(set2, ...)
или **set1 |= set2 | ...**

Объединяет все множества.
Результат сохраняет в set1

```
set1 = {1, 2}
set1.update({4, 5}, {3})
set1 |= {4, 5} | {3}
```

ooo

```
{1,2,3,4,5}
{1,2,3,4,5}
```

set1.intersection_update(set2, ...)
или **set1 &= set2 & ...**

Пересекает все множества.
Результат сохраняет в set1

```
set1 = {1, 2}
set1.intersection_update({1,5}, {1,3})
set1 &= {1,5} & {1,3}
```

ooo

```
{1}
{1}
```

set1.difference_update(set2, ...)
или **set1 -= set2 - ...**

Разность множеств. Результат сохраняет в set1

```
set1= {1, 2, 3, 4, 5}
set1.difference_update({1, 5}, {3})
set1 -= {1,5} - {3}
```

ooo

```
{2,4}
{2,4}
```

set1.symmetric_difference_update(set2, ...)
или **set1 ^= set2 ^ ...**

Симметрическая разность множеств. Результат сохраняет в set1

```
set1 = {1, 2}
set1.symmetric_difference_update({3, 2})
set1 ^= {3, 2}
```

ooo

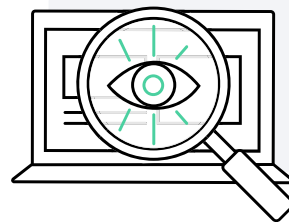
```
{1,3}
{1,3}
```

Демонстрация

Разберёмся, чем нам полезны множества



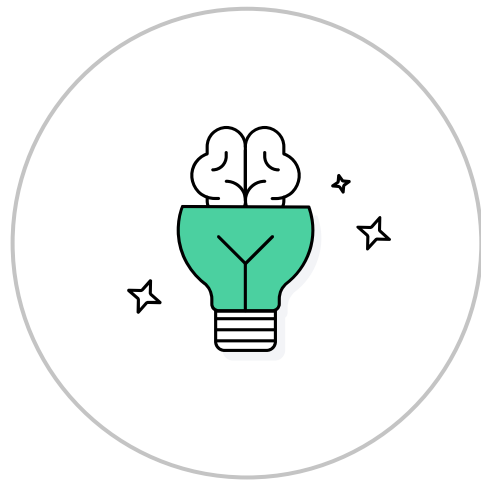
[Ссылка на код](#)



Кто быстрее

Попробуйте угадать:

- сколько времени займёт сравнение двух **списков** по 10 тыс. элементов?
- сколько времени займёт сравнение двух **множеств** по 10 тыс. элементов?

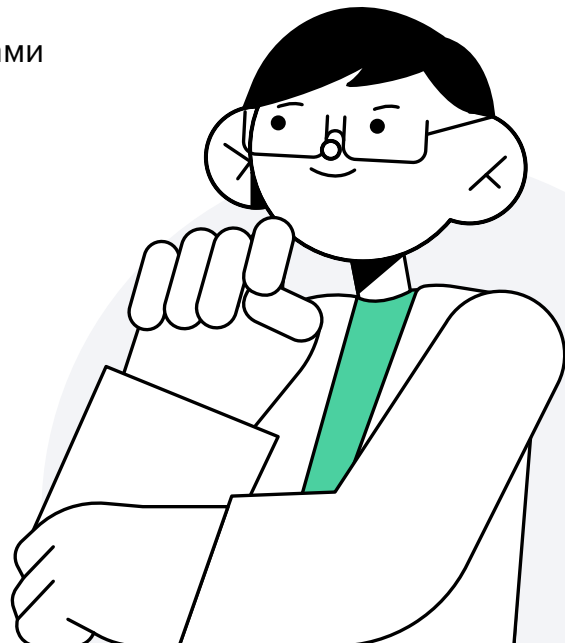


[Ссылка на код](#)

Итоги

Сегодня мы:

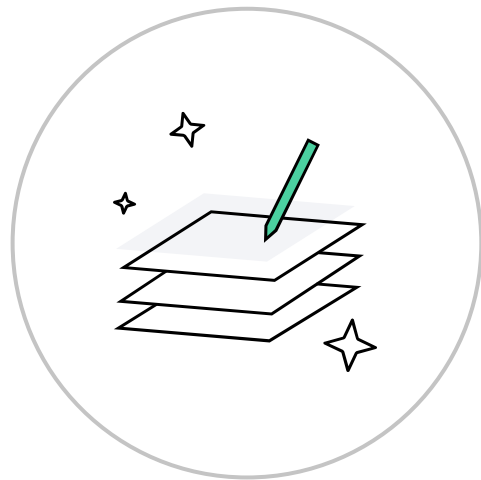
- 1 Узнали, для чего предназначены множества и чем они отличаются от списков
- 2 Научились работать с множествами в Python
- 3 Рассмотрели несколько вариантов записи операций над множествами
- 4 Поставили небольшой эксперимент по скорости обработки данных списками и множествами



Домашнее задание

Чтобы проверить, как вы поняли тему лекции,
и улучшить практические навыки программирования,
выполните **задания в тренажёре** в личном кабинете.

Вопросы по домашней работе задавайте в чате группы



Задавайте вопросы и пишите отзыв о лекции

Александр Бардин

