Sets

Множества в математике

Интересно: https://www.youtube.com/watch?v=jP3ceURvIYc

В математике множество – совокупность объектов, понимаемых как единое целое.

При этом предполагается, что объекты данной совокупности можно отличать друг от друга и от объектов, не входящих в эту совокупность. Например, можно говорить:

- о множестве всех студентов данного курса;
- множестве всех языков программирования;
- множестве всех натуральных чисел;
- множестве всех точек данного отрезка.

Студенты данного курса, языки программирования, натуральные числа, точки данного отрезка – элементы соответствующих множеств.

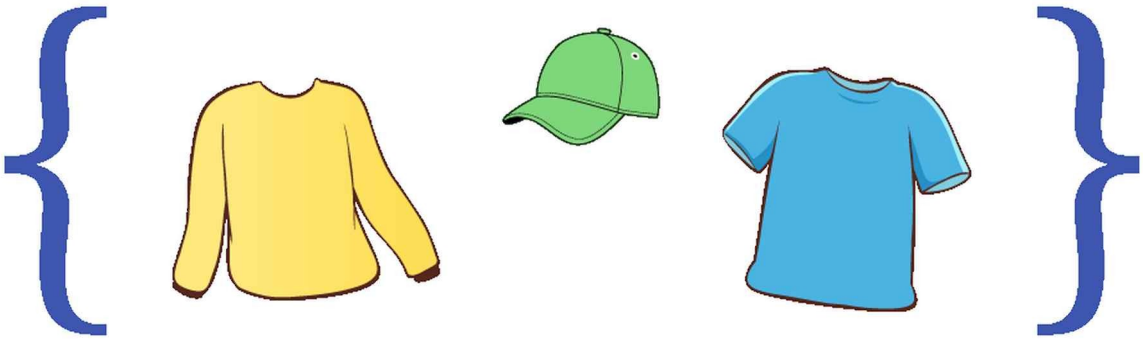
Обозначения

Обычно множества обозначают большими латинскими буквами: X,Y,…, а элементы множеств – латинскими строчными буквами: x,y,….

Запись $x \in X$($x \notin X$) означает, что x является (не является) элементом множества X.

Элементы множества указываются в фигурных скобках.

Рассмотрим множество вещей: {худи,кепка,футболка}. Такое множество содержит три элемента.



Конечные и бесконечные множества

Рассмотрим три множества:

- A={a,b,c,…,z} – множество букв английского алфавита;
- B={Тимур,Руслан,Роман,Оля} – множество имён авторов данного курса;

- N={1,2,3,4,5,…} – множество натуральных чисел.

Первые два множества содержат конечное количество элементов и являются конечными множествами. Третье же множество содержит бесконечно много элементов, поэтому и называется бесконечным множеством.

Мы используем символ …, чтобы показать, что элементы множества продолжаются. Другими словами, символ … означает "и так далее". Символ … можно использовать как в конечных, так и в бесконечных множествах.

Равенство множеств

Если два множества X и Y состоят из одних и тех же элементов, то они называются равными X=Y.

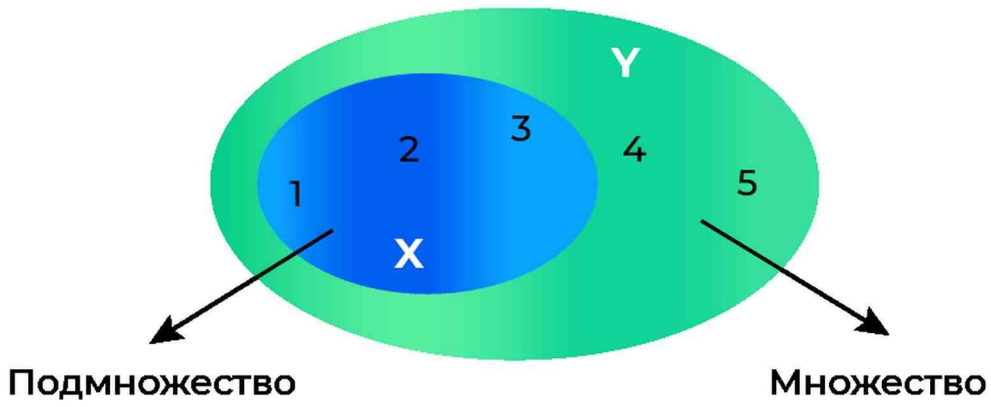
Например, если X ={a,b,c,d} и Y ={b,d,c,a}, то X=Y. Обратите внимание, порядок расположения элементов в записи множеств при их сравнении во внимание не принимается.

Еще один пример: X={1,2,3,4,5} и Y={множестванатуральныхчиселменьших6}. Очевидно, такие множества содержат абсолютно одинаковые элементы, поэтому равны.

Подмножество и надмножество

Если все элементы множества X принадлежат также и множеству Y, то говорят, что X является **подмножеством** Y, а записывается это так: X ⊂Y.

Например, X={1,2,3},Y={1,2,3,4,5}. Так как все элементы множества X содержатся в множестве Y, то мы говорим, что множество X является подмножеством множества Y.



Множество X ={1,2,3,6} не является подмножеством множества Y={1,2,3,4,5}, так как элемент 6 не содержится в Y.

Если множество X является подмножеством множества Y, то также говорят, что множество Y является **надмножеством** множества X, а записывается это так: Y ⊃X.

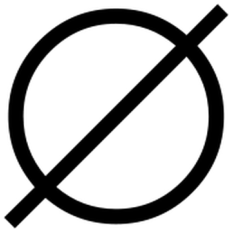
Заметим, что любое множество также является подмножеством самого себя. Про такое подмножество говорят **нестрогое подмножество:**

- множество {1,2,3} является **нестрогим** подмножеством множества {1,2,3};

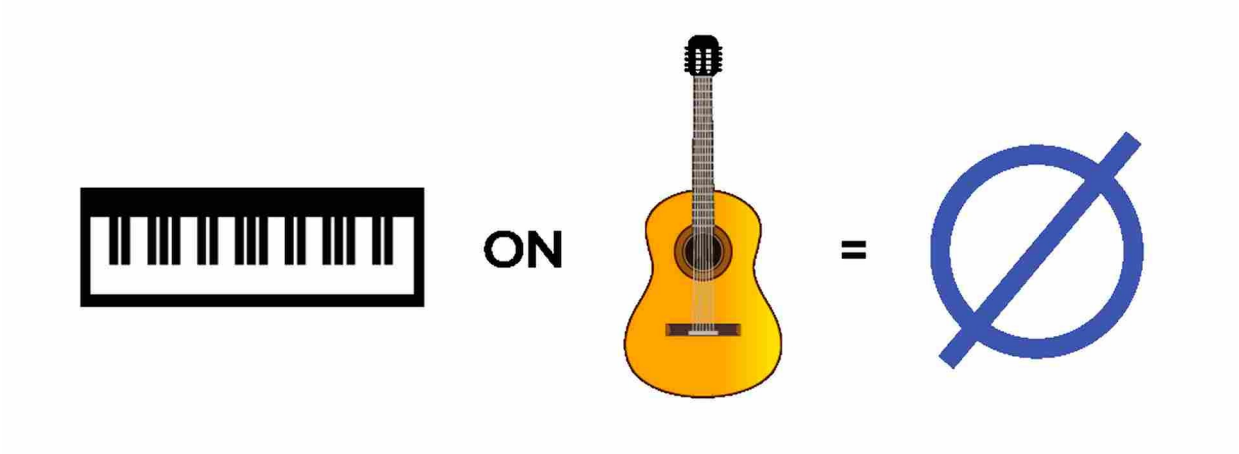
- множество $\{1,2,3\}$ является **строгим** подмножеством множества $\{1,2,3,4\}$.

Пустое множество

Для удобства работы с множествами и записи с их помощью различных математических высказываний, вводится понятие множества, не содержащего ни одного элемента. Оно называется пустым множеством и обозначается $\varnothing$.



Рассмотрим множество клавиш пианино, находящихся на гитаре. Очевидно такое множество не содержит элементов и является пустым.



Примеры пустых множеств:

- множество лошадей, пасущихся на луне;
- множество точек пересечения двух параллельных прямых на плоскости;
- множество квадратных уравнений, имеющих больше двух корней (действительных);
- множество людей, не любящих данный курс (ха-ха).

Пустое множество является подмножеством любого множества.

Примечания

Примечание 1. Множества – неупорядоченные совокупности, то есть, неважно, в каком порядке указаны элементы множества.

Примечание 2. Если множество X конечно, то через $|X|$ обозначается количество элементов множества X.

Примечание 3. Если множество X содержит n элементов, то оно имеет 2n подмножеств, включая пустое множество. Например, множество X={a,b,c} содержит 3 элемента и имеет 8 подмножеств:

1. {∅};
2. {a};
3. {b};
4. {c};
5. {a,b};
6. {a,c};
7. {b,c};
8. {a,b,c}.

Примечание 4. Раздел математики, занимающийся множествами, называется теорией множеств. Возникла эта теория во второй половине XIX века, главным образом в трудах немецкого математика Г. Кантора. Кантор определял множество как "любое собрание определенных и различимых между собой объектов нашей интуиции или интеллекта, мыслимое как единое целое".

Числовые множества.

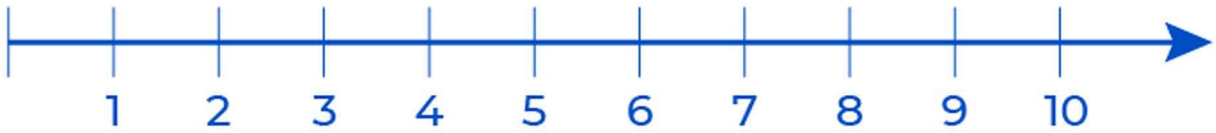
К основным числовым множествам математики относятся:

- множество натуральных чисел;
- множество целых чисел;
- множество рациональных чисел;
- множество вещественных чисел;
- множество комплексных чисел.

Натуральные числа

Исторически первыми появились натуральные числа, предназначенные для подсчёта материальных объектов (овец, кур, монет и т.д.). Множество натуральных чисел обозначается буквой N и содержит следующие числа:

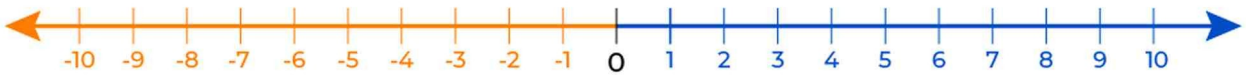
$N=\{1, 2, 3, 4, 5,\ldots\}$.



Обратите внимание, число ноль не является натуральным числом.

Целые числа

Если к множеству натуральных чисел N присоединить те же числа с противоположным знаком и ноль, то получится множество целых чисел. Множество целых чисел обозначается буквой Z и содержит следующие числа: $Z=\{0,\pm1,\pm2,\pm3,\pm4,\pm5,\ldots\}$. Множество натуральных чисел является подмножеством множества целых чисел, поскольку каждый элемент множества N принадлежит множеству Z.



Рациональные числа

Рациональным числом в математике называется любое число, представимое в виде частного двух целых чисел с отличным от нуля знаменателем. Множество рациональных чисел обозначается буквой Q и содержит следующие числа: $Q=\{m/n, m\in Z, n \in N\}$.

Множество целых чисел является подмножеством множества рациональных чисел, так как любое целое число можно представить в виде дроби со знаменателем, равным 1.

Любое рациональное число это либо конечная, либо бесконечная периодическая десятичная дробь. К примеру:

- 1/2 = 0.5 – конечная непериодическая дробь;
- 3/8 = 0.375 – конечная непериодическая дробь;
- 1/3 = 0.(3) – бесконечная периодическая десятичная дробь;
- 7/11 = 0.(63) – бесконечная периодическая десятичная дробь.

Иррациональные числа

Не все числа в математике можно представить в виде рационального числа. Примером служат числа:

- $\sqrt{2} \approx 1.414213562\ldots;$
- $\pi \approx 3.1415926535\ldots;$
- $e \approx 2.71828182845\ldots.$

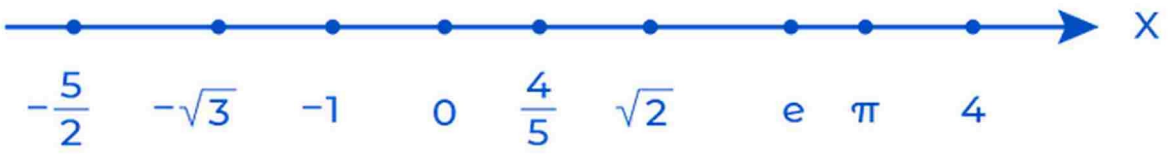
Такие числа называются иррациональными и являются бесконечными непериодическими дробями. Иными словами, в «бесконечных хвостах» иррациональных чисел нет никакой закономерности. Иррациональные числа часто обозначают буквой **I**

Вещественные числа

Объединение рациональных и иррациональных чисел образует множество вещественных чисел. Множество вещественных чисел R определяется так:

R=Q ∪ I

Геометрическая интерпретация множества вещественных чисел – это числовая прямая:



Каждому вещественному числу соответствует определённая точка числовой прямой, и наоборот – каждой точке числовой прямой обязательно соответствует некоторое вещественное число.

Множество вещественных чисел также называют множеством действительных чисел.

Множество комплексных чисел

Комплексным числом в математике называется любое число, представимое в виде

a + bi , где

a и b – вещественные числа, a i=sqrt(-1) мнимая единица. Множество комплексных чисел обозначается буквой C.

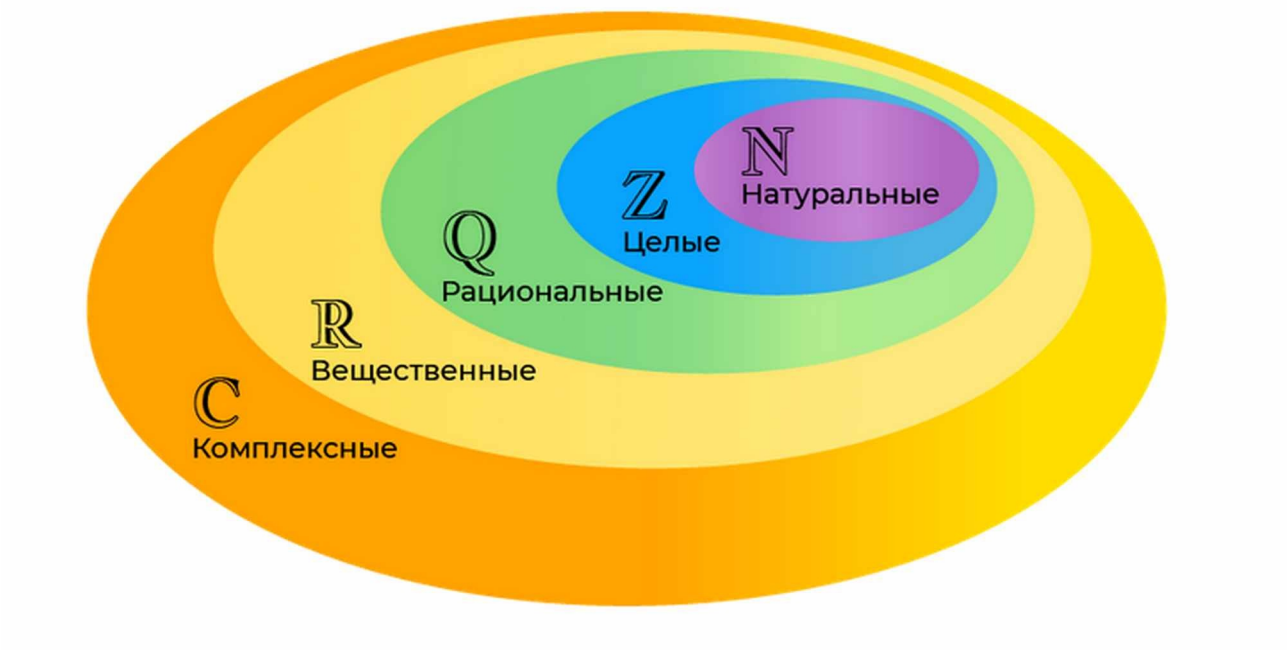
Множество вещественных чисел является подмножеством множества комплексных чисел, так как любое вещественное число можно представить в виде

a + bi , где b =0

Графическая иллюстрация числовых множеств

Несложно заметить, что каждое следующее множество является надмножеством предыдущего множества, так как содержит все его элементы:

$$N \subset Z \subset Q \subset R \subset C.$$



Числовые множества при решении уравнений

Рассмотрим алгебраические уравнения, корнями которых являются натуральные, целые, рациональные, вещественные и комплексные числа:

Уравнение	Корни	Множество чисел	Обозначение
$x - 3 = 0$	$x = 3$	Натуральные числа	$\mathbb{N}$
$x + 7 = 0$	$x = -7$	Целые числа	$\mathbb{Z}$
$4x - 1 = 0$	$x = \frac{1}{4}$	Рациональные числа	$\mathbb{Q}$
$x^2 - 2 = 0$	$x = \pm\sqrt{2}$	Вещественные числа	$\mathbb{R}$
$x^2 + 1 = 0$	$x = \pm i$	Комплексные числа	$\mathbb{C}$

Операции над множествами, диаграммы Эйлера-Венна

Диаграммы Эйлера-Венна

Диаграммы Эйлера-Венна – геометрическое представление множеств. Большой прямоугольник представляет универсальное множество U, а круги в нем – отдельные множества. Круги пересекаются в соответствии с условиями задачи. Точки внутри областей диаграммы — элементы соответствующих множеств. На диаграмме можно заштриховать образованные при пересечении кругов множества.

Операции над множествами

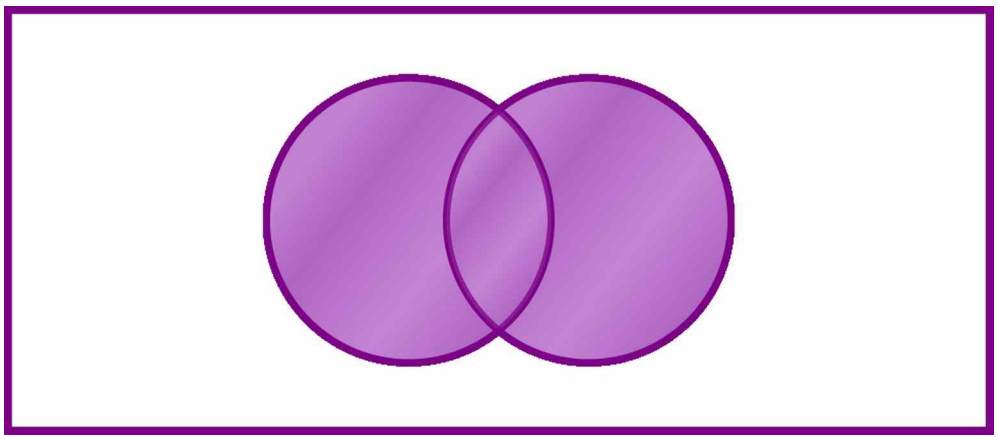
Операции над множествами выполняются для получения новых множеств из уже существующих.

Основные операции над множествами:

- объединение;
- пересечение;
- разность;
- симметрическая разность;
- дополнение.

Объединение множеств

Объединение множеств – множество, состоящее **из элементов, принадлежащих хотя бы одному** из объединяемых множеств.



Для объединения множеств используется символ **U**.

Например, если

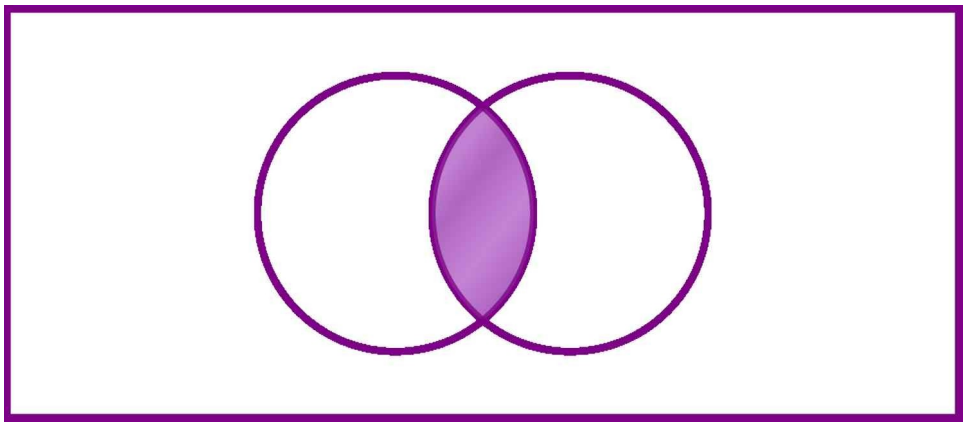
X={1,2,3,4,5}, Y={3,4,7,8,9}, то

X U Y={1,2,3,4,5,7,8,9}

Часто операцию объединения множеств отождествляют с операцией сложения.

Пересечение множеств

Пересечение множеств – множество, состоящее из элементов, **принадлежащих одновременно каждому** из пересекающихся множеств.



Для пересечения множеств используется символ ∩.

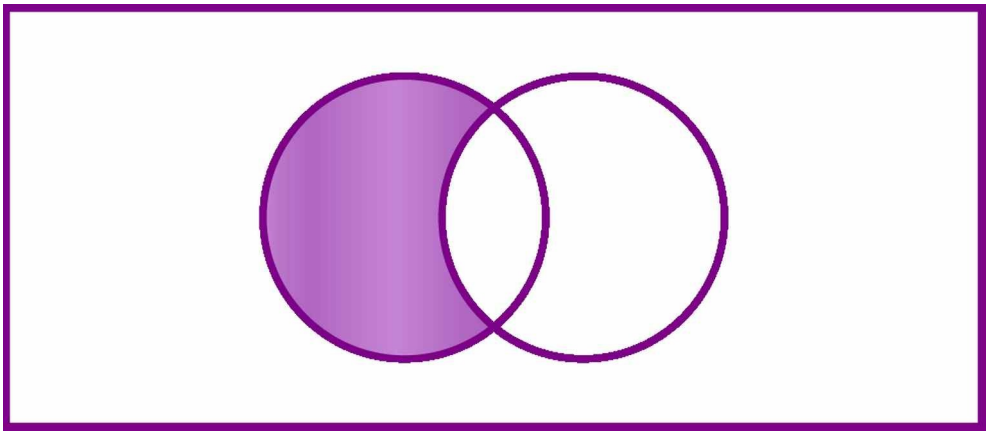
Например, если X={1,2,3,4,5},Y={3,4,7,8,9},

то X∩Y={3,4}.

Часто операцию **пересечения** множеств отождествляют с операцией **умножения**.

Разность множеств

Разность множеств – множество, в которое входят только элементы первого множества, не входящие во второе множество.



Для разности множеств используется символ \.

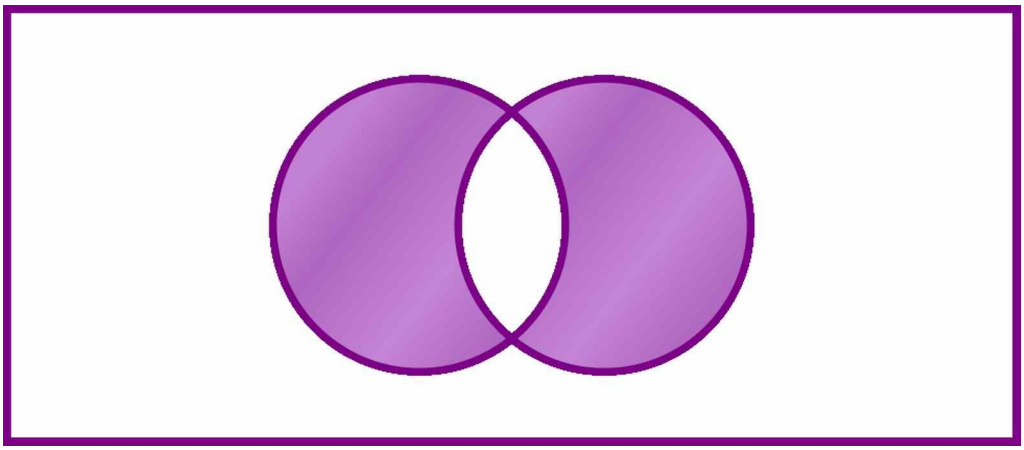
Например, если X={1,2,3,4,5},Y={3,4,7,8,9},

то X \ Y={1,2,5}.

Симметрическая разность

Симметрическая разность множеств – множество, включающее все элементы исходных множеств, не принадлежащие одновременно обоим исходным множествам.

Другими словами, симметрическая разность это множество $(X \cup Y) \setminus (X \cap Y)$.



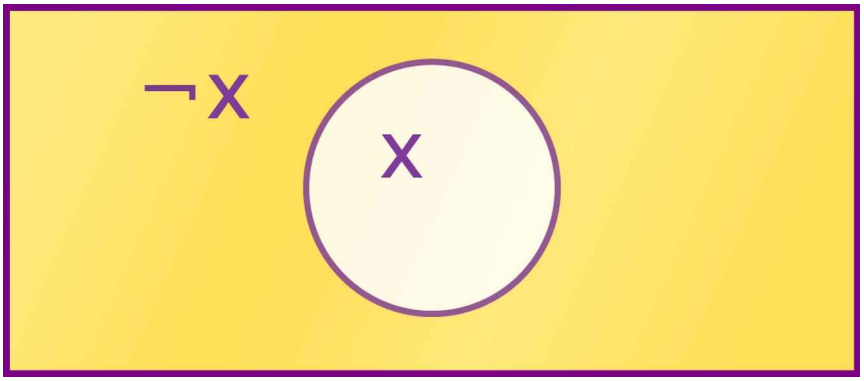
Для симметрической разности множеств используется символ $\triangle$.

Например, если $X=\{1,2,3,4,5\}, Y=\{3,4,7,8,9\}$,

то $X \triangle Y=\{1,2,5,7,8,9\}$.

Дополнение

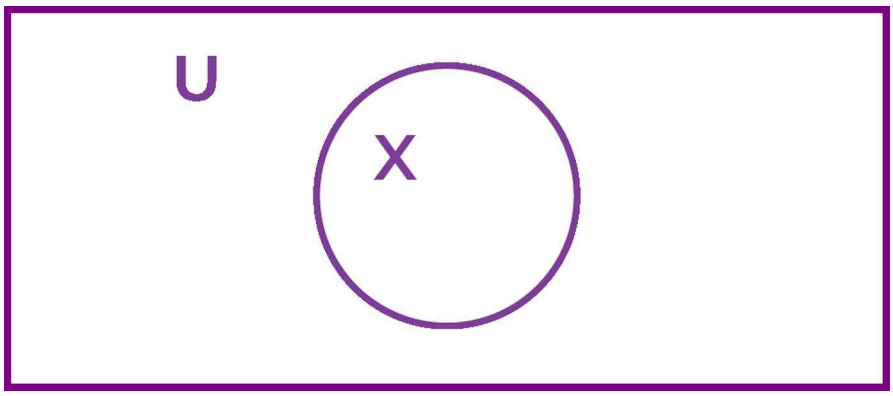
Дополнение множества – множество всех элементов, в нем не содержащихся.



Для операции дополнения множества используется символ ¬.

Примечания

Примечание 1. Предположим, что изучается некоторая область знаний. Множество всех элементов исследуемой области называется универсальным. На диаграммах универсальное множество обычно изображается множеством точек некоторого прямоугольника плоскости, а принадлежащие ему подмножества – кругами внутри прямоугольника.


U
X

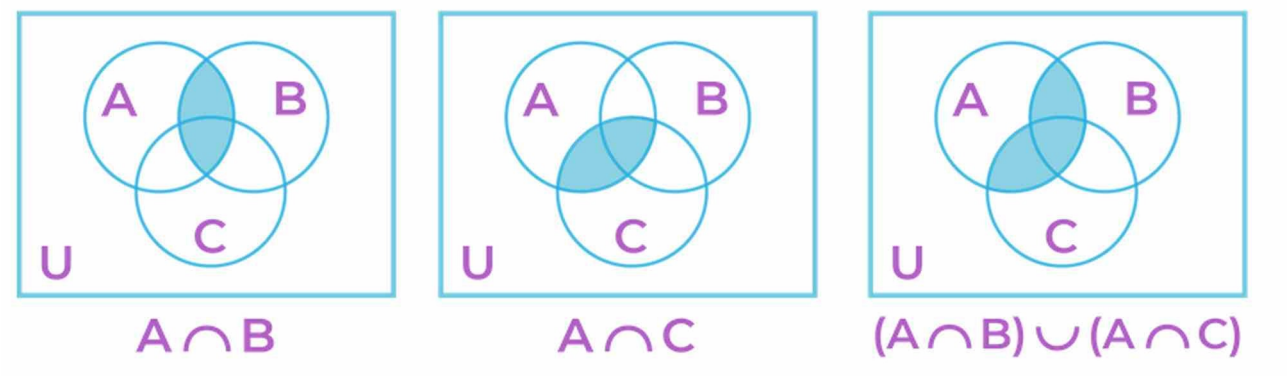
Диаграммы Эйлера-Венна при решении задач

Диаграммы Эйлера-Венна применяют для доказательства формул, решения текстовых задач и во многих других случаях.

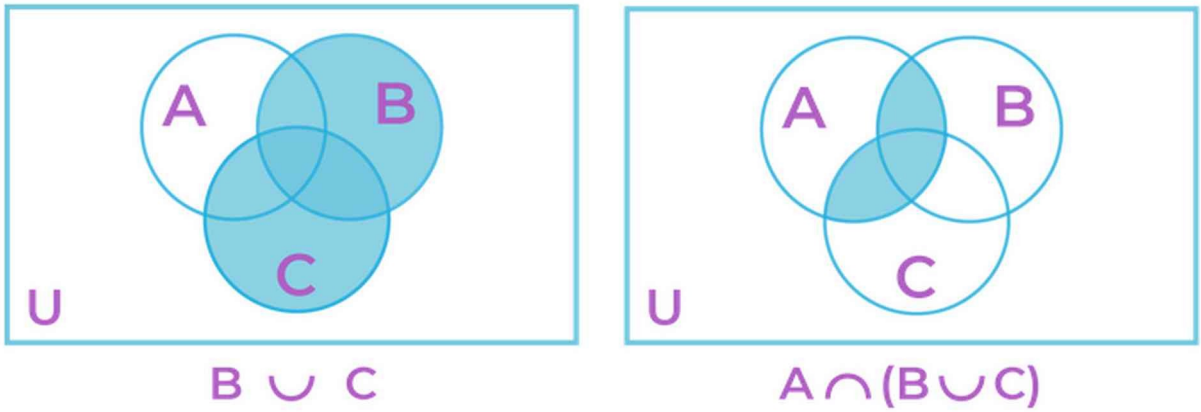
Задача 1. Докажите формулу $(A \cap B) \cup (A \cap C) = A \cap (B \cup C)$.

Решение. Используя диаграмму Эйлера-Венна, покажем, что обеим частям равенства соответствуют одна и та же область.

Левая часть:



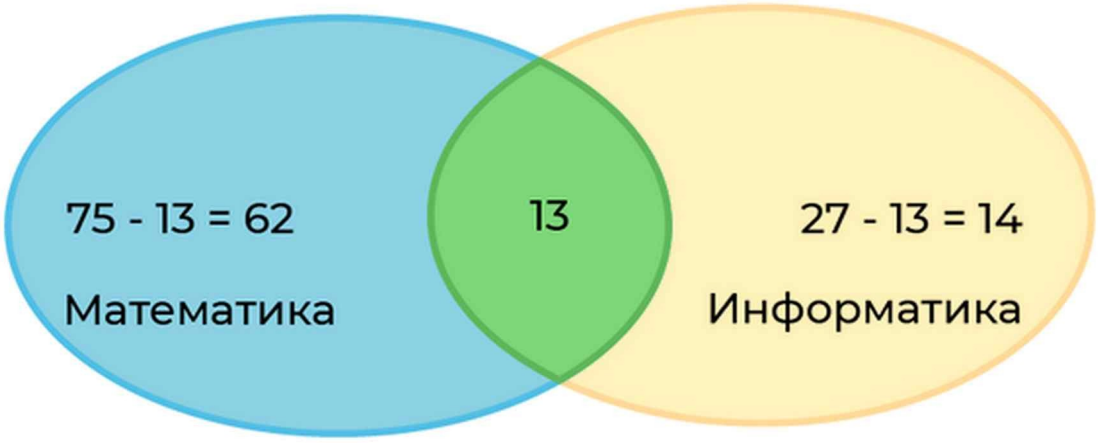
Правая часть:



Так как левой и правой частям формулы соответствует одна и та же область на диаграмме Эйлера-Венна, то формула верна.

Задача 2. Каждый ученик онлайн-школы BEEGEEK изучает или математику или информатику, или и то и другое одновременно. Всего 75 учеников изучает математику, а 27 – информатику и только 13 – оба предмета. Сколько учеников учится в онлайн-школе BEEGEEK?

Решение. Введем обозначения: множество учеников, изучающих математику – М, информатику – И. Изображаем множества на диаграмме Эйлера-Венна в наиболее общем случае.



Рассуждаем следующим образом: оба предмета изучают 13 учеников. Значит только математику изучают 75−13=62 ученика, только информатику изучают 27−13=14 ученика.

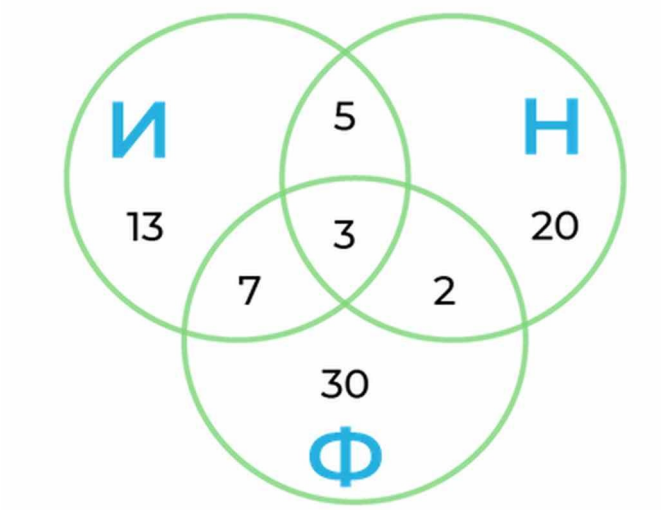
Таким образом всего в школе учится 62+13+14=89 учеников.

Ответ: 89.

Задача 3. Опрос 100 студентов дал следующие результаты по количеству изучающих разные иностранные языки: испанский – 28, немецкий – 30, французский – 42, испанский и немецкий – 8, испанский и французский – 10, немецкий и французский – 5, все три языка – 3. Ответьте на вопросы:

1. Сколько студентов не изучает ни одного языка?
2. Сколько студентов изучает один французский язык?

Решение. Введем обозначения: множество студентов, изучающих немецкий язык – Н, французский – Ф, испанский – И. Изображаем множества на диаграмме Эйлера-Венна в наиболее общем случае.



Рассуждаем следующим образом: все три языка изучают 3 студента. Значит, одновременно изучают только немецкий и французский 5−3=2, одновременно изучают только немецкий и испанский 8−3=5, одновременно изучают только французский и испанский 10−3=7.

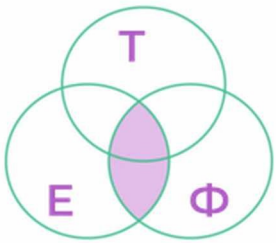
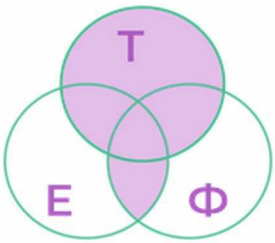
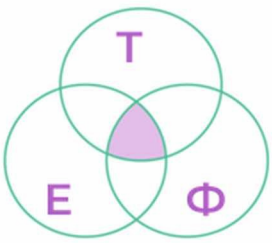
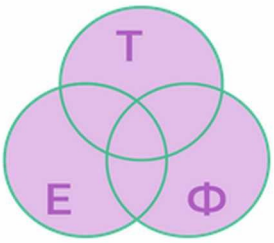
Поскольку всего 28 студентов изучают испанский, то число студентов изучающих только испанский язык 28−3−7−5=13. Аналогично находим число студентов изучающих только немецкий язык 30−3−2−5=20, число студентов изучающих только французский язык 42−3−7−2=30. Находим число студентов, изучающих иностранные языки 13+20+30+3+5+7+2=80. Таким образом, число студентов не изучающих язык, 100−80=20.

Ответ. 20 студентов не изучают ни одного языка, 30 студентов изучает только французский язык.

Задача 4. В языке запросов поискового сервера для обозначения логической операции «ИЛИ» используется символ «|», а для обозначения логической операции «И» — символ «&». Приведены запросы к поисковому серверу. Для каждого запроса указан его код — соответствующая буква от А до Г. Расположите коды запросов слева направо в порядке возрастания количества страниц, которые нашел поисковый сервер по каждому запросу.

Код	Запрос
А	Есенин & Фет
Б	(Есенин & Фет) | Тютчев
В	Есенин & Фет & Тютчев
Г	Есенин | Фет | Тютчев

Решение. Изобразим диаграммы Эйлера-Венна для всех запросов:

А	Б	В	Г
Есенин & Фет	(Есенин & Фет) | Тютчев	Есенин & Фет & Тютчев	Есенин | Фет | Тютчев
			

Ответ: ВАБГ.

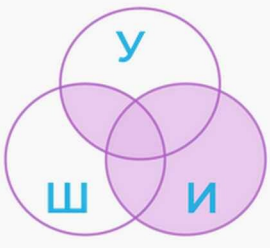
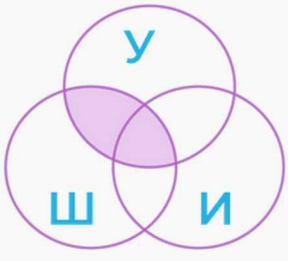
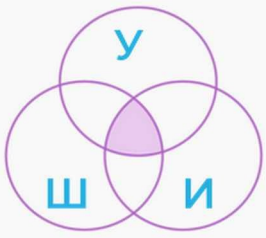
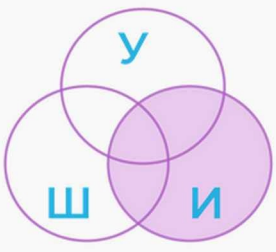
Задача 5. В таблице приведены запросы и количество найденных по ним страниц некоторого сегмента сети Интернет.

Запрос	Найдено страниц (в тысячах)
Уэльс & Шотландия | Ирландия	450
Уэльс & Шотландия	213
Уэльс & Шотландия & Ирландия	87

Какое количество страниц (в тысячах) будет найдено по запросу **Ирландия**?

Решение. Введем обозначения: Уэльс – У, Шотландия – Ш, Ирландия – И. Изображаем множества на диаграмме Эйлера-Венна в наиболее общем случае.

Отметим известные и неизвестные составные высказывания

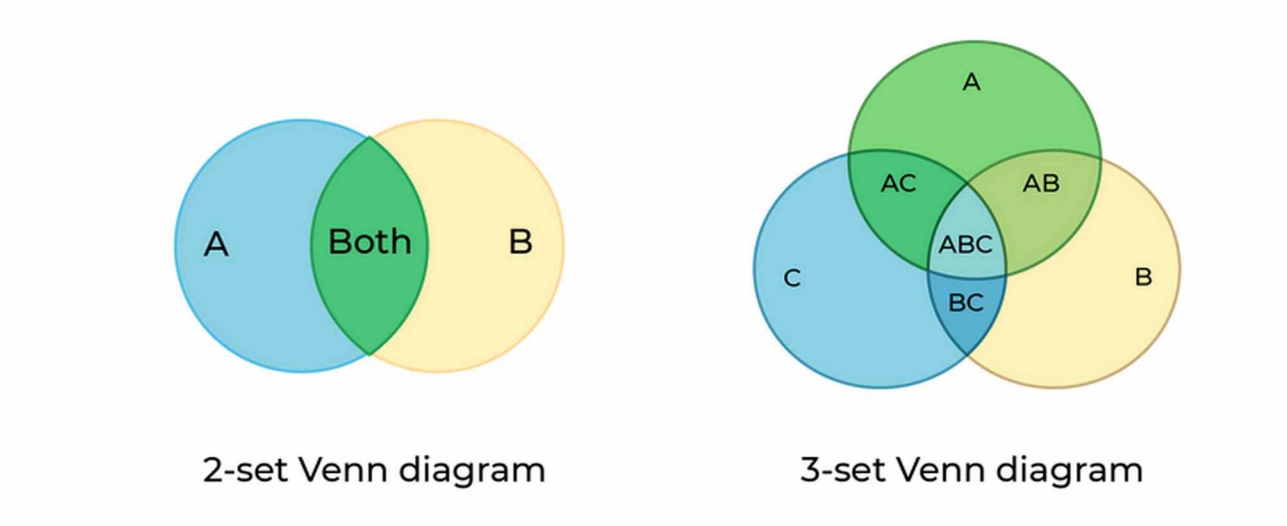
Уэльс & Шотландия | Ирландия	Уэльс & Шотландия	Уэльс & Шотландия & Ирландия	Ирландия
			

Итак, по запросу Ирландия будет найдено 450−213+87=324 страниц.

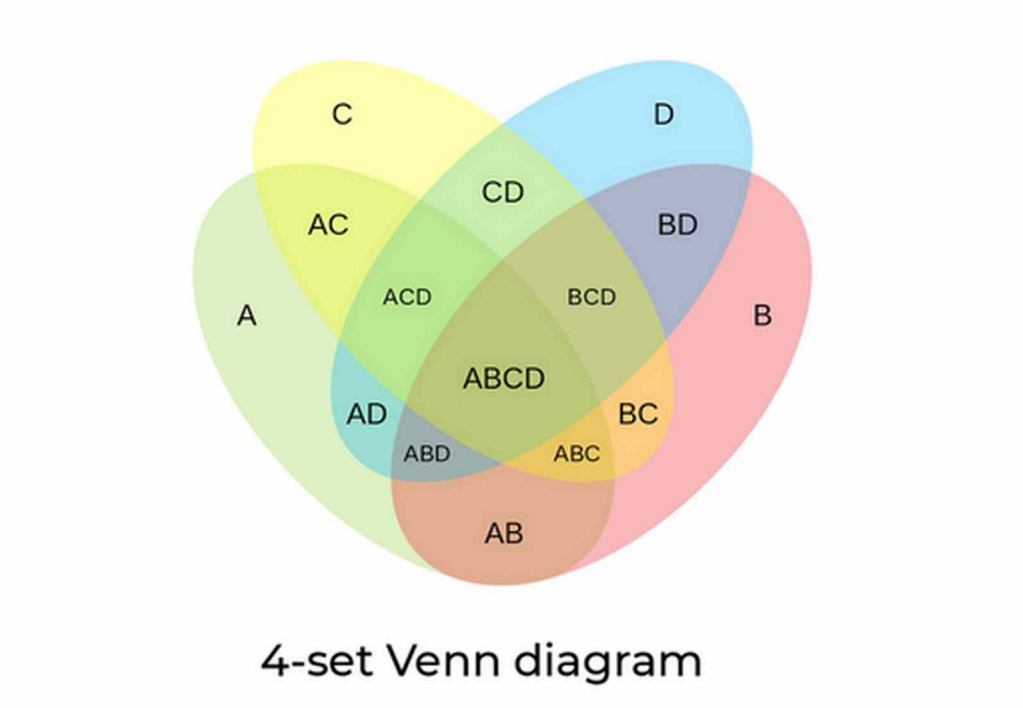
Ответ: 324.

Примечания

Примечание 1. Диаграммы Эйлера-Венна для 2,3 множеств выглядят так:



Примечание 2. Диаграмма Эйлера-Венна для 4 множеств выглядят так:



Для 5 и более множеств диаграммы Эйлера-Венна неудобны.

Задачи

Задача 1.

Множества A и B содержат 5 и 6 элементов соответственно, а множество $A \cap B$ – 2 элемента. Сколько элементов в множестве $A \cup B$?

Введите численный ответ

Всё получилось!

Верно решили **13 933** учащихся
Из всех попыток **58%** верных

9

Задача 2.

Каждый ученик в классе изучает английский или французский язык. Английский язык изучает 25 человек, французский – 27, а оба языка – 18. Сколько учащихся в классе?

Введите численный ответ

Хорошие новости, верно!

Верно решил **13 881** учащийся
Из всех попыток **80%** верных

34

Задача 3.

В одном из классов онлайн-школы BEEGEEK учится 67 человек. Из них 47 умеют решать задачи с параметрами, 35 –экономические задачи, а 23 – и те и другие. Сколько человек в классе не умеют решать ни экономические задачи, ни задачи с параметрами?

Введите численный ответ

Верно.

Верно решили **13 785** учащихся
Из всех попыток **68%** верных

8

Задача 4.

В классе учатся 30 учеников. Среди них 17 отличников по математике, 10 отличников по физике и 13 — по информатике. Трое — отличники по всем предметам, пятеро — по математике и физике, четверо — по физике и информатике, а 6 человек — по математике и информатике. Сколько учеников не являются отличниками ни по одному из этих предметов?

Введите численный ответ

Отлично!

Верно решили **13 320** учащихся
Из всех попыток **37%** верных

2

Задача 5.

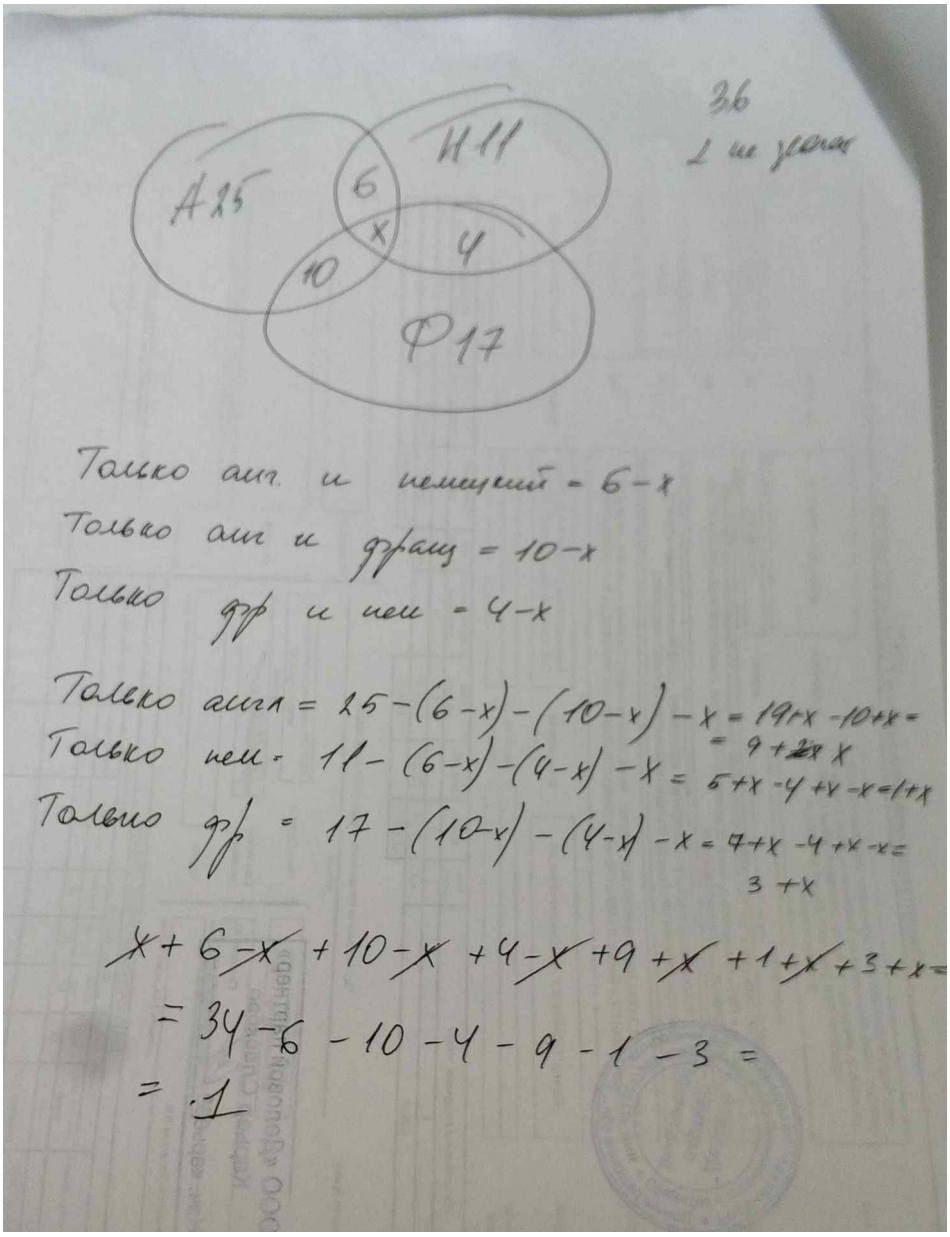
В классе 36 учеников, из которых двое не знают иностранных языков. Английским языком владеют 25 учеников, немецким — 11, французским — 17 человек. Известно, что и английским, и немецким языком владеют 6 учеников, и английским, и французским — 10 учеников, и немецким, и французским — 4 ученика. Сколько учеников владеют всеми тремя языками?

Введите численный ответ

Абсолютно точно.

Верно решил **12 961** учащийся
Из всех попыток **46%** верных

1



Введение в множества python

Аннотация. Начинаем изучение множеств в Python (тип данных `set`). Этот тип данных аналогичен математическим множествам, он поддерживает быстрые операции проверки наличия элемента в множестве, добавления и удаления элементов, операции объединения, пересечения и многие другие.

Множества

В прошлых уроках мы изучили три типа коллекций в Python:

- списки – изменяемые коллекции элементов;
- строки – неизменяемые коллекции символов;
- кортежи – неизменяемые коллекции элементов.

Следующий тип коллекций (наборов данных) – **множество**.

Множество – структура данных, организованная так же, как математические множества.

Важно знать:

- все элементы множества различны (уникальны), два элемента не могут иметь одинаковое значение;
- множества неупорядочены, то есть элементы не хранятся в каком-то определенном порядке;
- **элементы множества должны относиться к неизменяемым типам данных**;
- хранящиеся в множестве элементы могут иметь разные типы данных.

Структура данных (data structure) — программная единица, позволяющая **хранить и обрабатывать** множество однотипных и/или логически связанных данных.

Создание множества

Чтобы создать множество, нужно перечислить его элементы через запятую в фигурных скобках:

```
numbers = {2, 4, 6, 8, 10}
languages = {"Python", "C#", "C++", "Java"}
```

Множество `numbers` состоит из 5 элементов, и каждый из них — **целое число**.

Множество `languages` состоит из 4 элементов, каждый из которых — **строка.**

Множества могут содержать значения **разных типов данных**:

```
info = {'Timur', 1992, 61.5}
```

Множество `info` содержит строковое значение, целое число и число с плавающей точкой.

Не создавайте переменные с именем `set`. Это очень плохая практика.

Пустое множество

Создать пустое множество можно с помощью встроенной функции, которая называется `set()`:

```
myset = set()  # пустое множество
```

Обратите внимание — создать пустое множество с помощью пустых фигурных скобок нельзя:

```
myset = {}  # создается словарь
```

С помощью пустых фигурных скобок создаются словари: так сложилось исторически. Дело в том, что словари появились в Python раньше, чем множества.

Пустое множество создаётся исключительно через `set()`.

Вывод множества

Для вывода всего множества можно использовать функцию `print()`:

```
numbers = {2, 4, 6, 8, 10}
languages = {"Python", "C#", "C++", "Java"}
mammals = {"cat", "dog", "fox", "elephant"}

print(numbers)
print(languages)
print(mammals)
```

Функция `print()` выводит на экран элементы множества в фигурных скобках, разделенные запятыми:

```
{2, 4, 6, 8, 10}
{"C#", "Python", "Java", "C++"}
{"dog", "cat", "fox", "elephant"}
```

Обратите внимание: при выводе множества порядок элементов может отличаться от существовавшего при его создании, поскольку множества — неупорядоченные коллекции данных.

Встроенная функция set()

Встроенная функция `set()` помимо создания пустого множества может преобразовывать некоторые типы объектов в множества.

В функцию `set()` можно передать один аргумент. Передаваемый аргумент должен быть итерируемым объектом, таким как список, кортеж или строковое значение. Отдельные элементы объекта, передаваемого в качестве аргумента, становятся элементами множества:

```
myset1 = set(range(10))          # множество из элементов последовательности
myset2 = set([1, 2, 3, 4, 5])    # множество из элементов списка
myset3 = set('abcd')             # множество из элементов строки
myset4 = set((10, 20, 30, 40))   # множество из элементов кортежа
```

Пустое множество также можно создать передав функции `set()` в качестве аргумента пустой список, строку или кортеж:

```
emptyset1 = set([])  # пустое множество из пустого списка
emptyset2 = set('')  # пустое множество из пустой строки
emptyset3 = set(())  # пустое множество из пустого кортежа
```

Дубликаты при создании множеств

Множества не могут содержать повторяющиеся элементы. Если в функцию `set()` передать аргумент, содержащий повторяющиеся элементы, то в множестве появится только один из этих повторяющихся элементов.

Приведенный ниже код:

```
myset1 = {2, 2, 4, 6, 6}
myset2 = set([1, 2, 2, 3, 3])
myset3 = set("aaaaabbbbccccddd")

print(myset1)
print(myset2)
print(myset3)
```

выводит (порядок элементов может отличаться):

```
{2, 4, 6}
{1, 2, 3}
{"b", "c", "d", "a"}
```

Если требуется создать множество, в котором каждый элемент — строковое значение, содержащее более одного символа, то используем код:

```
myset = set(['aaa', 'bbbb', 'cc'])
print(myset)
```

Приведенный выше код выводит (порядок элементов может отличаться):

```
{'bbbb', 'aaa', 'cc'}
```

Если же создать множество следующим образом:

```
myset = set('aaa bbbb cc')
print(myset)
```

то мы получим (порядок элементов может отличаться):

```
{' ', 'c', 'a', 'b'}
```

Обратите внимание на наличие пробела в качестве элемента множества `myset`.

Примечания

Примечание 1. Элементы множества могут принадлежать любому **неизменяемому типу данных**: быть числами, строками, кортежами и т.д. Элементы изменяемых типов данных не могут входить в множества, в частности, нельзя сделать элементом множества список или

другое множество. Требование неизменяемости элементов множества накладывается особенностями представления множеств в Python.

Приведенный ниже код:

```
myset1 = {1, 2, [5, 6], 7}  # множество не может содержать список
myset2 = {1, 2, {5, 6}, 7}  # множество не может содержать множество
```

приводит к ошибке:

```
TypeError: unhashable type: 'list'
TypeError: unhashable type: 'set'
```

Однако приведенный ниже код:

```
myset = {1, 2, (5, 6), 7}  # множество может содержать кортеж
```

работает, как полагается.

Примечание 2. Документация по множествам доступна по ссылке.

Примечание 3. Отличная статья с хабра про множества.

Основы работы с множествами

Работа с множествами очень сильно напоминает работу со списками, поскольку и множества, и списки содержат отдельные элементы, хотя элементы множества уникальны, а списки могут содержать повторяющиеся элементы. Многое из того, что мы делали со списками, доступно и при работе со множествами.

Функция len()

Длиной множества называется количество его элементов. Чтобы посчитать длину множества, используют встроенную функцию len() (от слова length – длина).

Следующий программный код:

```
myset1 = {2, 2, 4, 6, 6}
myset2 = set([1, 2, 2, 3, 3, 4, 4, 5, 5])
myset3 = set('aaaaabbbbccccddd')
print(len(myset1))
print(len(myset2))
print(len(myset3))
```

выведет:

```
3
5
4
```

Оператор принадлежности in

Оператор in позволяет проверить, содержит ли множество некоторый элемент.

Рассмотрим следующий код:

```
numbers = {2, 4, 6, 8, 10}
if 2 in numbers:
    print('Множество numbers содержит число 2')
else:
    print('Множество numbers не содержит число 2')
```

Такой код проверяет, содержит ли множество numbers число 22 и выводит соответствующий текст:

```
Множество numbers содержит число 2
```

Мы можем использовать оператор in вместе с логическим оператором not. Например

```
numbers = {2, 4, 6, 8, 10}
if 0 not in numbers:
    print('Множество numbers не содержит нулей')
```

Оператор принадлежности in работает очень быстро на множествах – намного быстрее, чем на списках. Поэтому если требуется часто осуществлять поиск в коллекции уникальных данных, то множество – подходящий выбор.

Встроенные функции sum(), min(), max()

Встроенная функция sum() принимает в качестве аргумента множество чисел и вычисляет сумму его элементов.

Следующий программный код:

```
numbers = {2, 2, 4, 6, 6}
print('Сумма всех элементов множества =', sum(numbers))
```

выводит:

```
Сумма всех элементов множества = 12
```

Встроенные функции min() и max() принимают в качестве аргумента множество и находят минимальный и максимальный элементы соответственно.

Следующий программный код:

```
numbers = {2, 2, 4, 6, 6}
print('Минимальный элемент =', min(numbers))
print('Максимальный элемент =', max(numbers))
```

выводит:

```
Минимальный элемент = 2
Максимальный элемент = 6
```

Примечания

Примечание 1. Индексация и срезы недоступны для множеств.

Примечание 2. Операция конкатенации + и умножения на число * недоступны для множеств.

Перебор элементов множества

Перебор элементов множества осуществляется точно так же, как и перебор элементов списка, то есть с помощью цикла `for`.

Для вывода элементов множества **каждого на отдельной строке** можно использовать следующий код:

```
numbers = {0, 1, 1, 2, 3, 3, 3, 5, 6, 7, 7}

for num in numbers:
    print(num)
```

Такой код выведет (порядок элементов может отличаться):

```
0
1
2
3
5
6
7
```

Мы также можем использовать операцию **распаковки множества**.

Приведенный ниже код:

```
numbers = {0, 1, 1, 2, 3, 3, 3, 5, 6, 7, 7}
print(*numbers, sep='\n')
```

выводит (порядок элементов может отличаться):

```
0
1
2
3
5
6
7
```

Не стоит забывать, что множества – неупорядоченные коллекции, поэтому полагаться на порядок вывода элементов не стоит. Если нужно гарантировать порядок вывода элементов (по возрастанию/убыванию), то необходимо воспользоваться встроенной функцией `sorted()`.

Приведенный ниже код:

```
numbers = {0, 1, 1, 2, 3, 3, 3, 5, 6, 7, 7}

sorted_numbers = sorted(numbers)
print(*sorted_numbers, sep='\n')
```

будет **гарантированно** выводить элементы множества в порядке возрастания.

Обратите внимание на то, что функция `sorted()` возвращает отсортированный список, а не множество. Не путайте встроенную функцию `sorted()` и списочный метод `sort()`. Множества **не содержат** метода `sort()`.

Сравнение множеств

Множества можно сравнивать между собой. Равные множества имеют одинаковую длину и содержат равные элементы. Для сравнения множеств используются операторы `==` и `!=`.

Приведенный ниже код:

```
myset1 = {1, 2, 3, 3, 3, 3}
myset2 = {2, 1, 3}
myset3 = {1, 2, 3, 4}

print(myset1 == myset2)
print(myset1 == myset3)
print(myset1 != myset3)
```

выводит:

```
True
False
True
```

Примечания

Примечание 1. Встроенная функция `sorted()` имеет опциональный параметр `reverse`. Если установить этот параметр в значение `True`, произойдет сортировка по убыванию.

Приведенный ниже код:

```
numbers = {0, 1, 1, 2, 3, 3, 3, 5, 6, 7, 7}

sortnumbers = sorted(numbers, reverse=True)
print(*sortnumbers, sep='\n')
```

гарантированно выводит:

```
7
6
5
3
2
1
0
```

Примечание 2. Код для работы с множествами нужно писать так, чтобы результат его выполнения не зависел от расположения элементов и был одинаковым при любом порядке обхода, последовательного обращения ко всем элементам.

Методы множеств. Часть 1

Добавление элементов. Метод add()

Мы научились создавать множества, элементы которых известны на этапе создания. Следующий шаг – научиться добавлять элементы в уже существующие множества.

Для добавления нового элемента в множество используется метод **`add()`**.

Следующий программный код:

```
numbers = {1, 1, 2, 3, 5, 8, 3}  # создаем множество

numbers.add(21)  # добавляем число 21 в множество
numbers.add(34)  # добавляем число 34 в множество

print(numbers)
```

выводит (порядок элементов может отличаться):

```
{1, 2, 3, 34, 5, 8, 21}
```

Не забывайте, что порядок элементов при выводе множества абсолютно произвольный.

Обратите внимание, для использования метода **`add()`** требуется предварительно созданное множество, при этом оно может быть пустым.

Следующий программный код:

```
numbers = set()  # создаем пустое множество

numbers.add(1)
numbers.add(2)
numbers.add(3)
numbers.add(1)

print(numbers)
```

выводит (порядок элементов может отличаться):

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

Если требуется внести несколько значений в множество, то можно воспользоваться циклом **`for`**.

Следующий программный код:

```
numbers = set()  # создаем пустое множество

for i in range(10):
    numbers.add(i * i + 1)

print(numbers)
```

выводит (порядок элементов может отличаться):

{1, 2, 65, 5, 37, 10, 17, 50, 82, 26}

Удаление элемента. Метод remove()

Для удаления элементов из множества используются методы:

- `remove()`;
- `discard()`;
- `pop()`.

Метод `remove()` удаляет элемент из множества с генерацией исключения (ошибки) в случае, если такого элемента нет.

Следующий программный код:

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

numbers.remove(3)
print(numbers)
```

выводит (порядок элементов может отличаться):

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

Следующий программный код:

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

numbers.remove(10)
print(numbers)
```

приводит к возникновению ошибки **`KeyError`**, так как элемент 10 отсутствует в множестве.

Метод discard()

Метод **`discard()`** удаляет элемент из множества без генерации исключения (ошибки), если элемент отсутствует.

Следующий программный код:

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

numbers.discard(3)
print(numbers)
```

выводит (порядок элементов может отличаться):

{1, 2, 4, 5}

Следующий программный код:

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

numbers.discard(10)
```

```
print(numbers)
```

не приводит к возникновению ошибки и выводит (порядок элементов может отличаться):

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

Метод pop()

Метод **`pop()`** удаляет и возвращает случайный элемент из множества с генерацией исключения (ошибки) при попытке удаления из пустого множества.

Рассмотрим программный код:

```
numbers = {1, 2, 3, 4, 5}
print('до удаления:', numbers)
num = numbers.pop()              # удаляет случайный элемент множества, возвращая его
print('удалённый элемент:', num)
print('после удаления:', numbers)
```

Результат работы такого кода случаен, например, такой код может вывести:

```
до удаления: {1, 2, 3, 4, 5}
удалённый элемент: 1
после удаления: {2, 3, 4, 5}
```

Метод `pop()` можно воспринимать как неконтролируемый способ удаления элементов по одному из множества.

Метод clear()

Метод **`clear()`** удаляет все элементы из множества.

Следующий программный код:

```
numbers = {1, 2, 3, 4, 5}
numbers.clear()

print(numbers)
```

выведет:

```
set()
```

В результате получили пустое множество.

Обратите внимание на то, что пустое множество выводится как `set()`, а не как `{}`. С помощью `{}` выводится пустой словарь.

Примечания

Примечание 1. Если мы не изменяли множество, порядок обхода элементов при помощи цикла `for` не изменится.

Примечание 2. После изменения множества (методы **`add()`**, **`remove()`**, и т.д.) порядок элементов может измениться произвольным образом.

Методы множеств. Часть 2

Операции над множествами

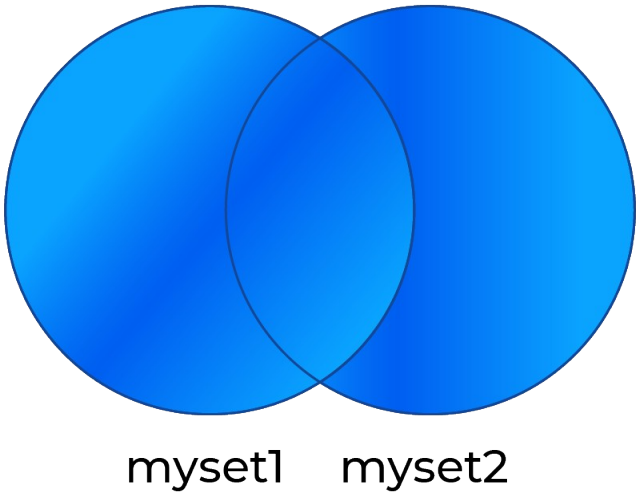
Основные операции над множествами:

- объединение множеств;
- пересечение множеств;
- разность множеств;
- симметрическая разность множеств.

Для каждой операции есть метод и оператор.

Объединение множеств: метод union()

Объединение множеств – это множество, состоящее из элементов, принадлежащих хотя бы одному из объединяемых множеств. Для этой операции существует метод **`union()`**.



Приведенный ниже код:

```
myset1 = {1, 2, 3, 4, 5}
myset2 = {3, 4, 6, 7, 8}
myset3 = myset1.union(myset2)
print(myset3)
```

выводит (порядок элементов может отличаться):

```
{1, 2, 3, 4, 5, 6, 7, 8}
```

Обратите внимание, метод **`union()` возвращает новое множество** в которое входят все элементы множеств `myset1` и `myset2`. Для изменения текущего множества используется метод **`update()`**.

Для объединения двух множеств можно также использовать оператор `|`.

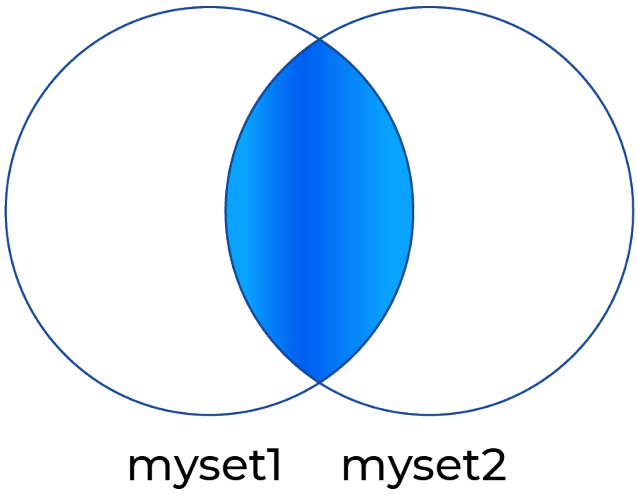
Результат выполнения приведенного ниже кода:

```
myset1 = {1, 2, 3, 4, 5}
myset2 = {3, 4, 6, 7, 8}

myset3 = myset1 | myset2
print(myset3)
```

аналогичен предыдущему.

Пересечение множеств: метод intersection()

Пересечение множеств – это множество, состоящее из элементов, принадлежащих одновременно каждому из пересекающихся множеств. Для этой операции существует метод **`intersection()`**.



Приведенный ниже код:

```
myset1 = {1, 2, 3, 4, 5}
myset2 = {3, 4, 6, 7, 8}

myset3 = myset1.intersection(myset2)
print(myset3)
```

выводит (порядок элементов может отличаться):

```
{3, 4}
```

Обратите внимание, метод **`intersection()`** возвращает новое множество в которое входят общие элементы множеств `myset1` и `myset2`. Для изменения текущего множества используется метод `intersection_update()`.

Для пересечения двух множеств можно также использовать оператор &.

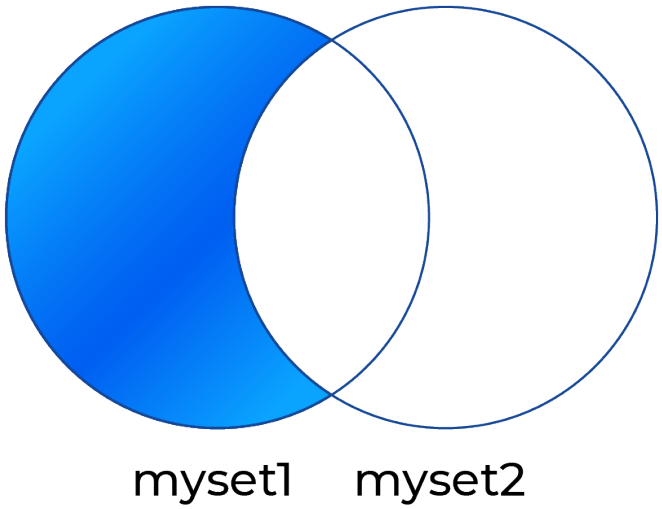
Результат выполнения приведенного ниже кода:

```
myset1 = {1, 2, 3, 4, 5}
myset2 = {3, 4, 6, 7, 8}

myset3 = myset1 & myset2
print(myset3)
```

аналогичен предыдущему.

Разность множеств: метод difference()

Разность множеств – это множество, в которое входят все элементы первого множества, не входящие во второе множество. Для этой операции существует метод **`difference()`**.



Приведенный ниже код:

```
myset1 = {1, 2, 3, 4, 5}
myset2 = {3, 4, 6, 7, 8}

myset3 = myset1.difference(myset2)
print(myset3)
```

выводит (порядок элементов может отличаться):

```
{1, 2, 5}
```

Для разности двух множеств можно также использовать оператор `-`.

Результат выполнения приведенного ниже кода:

```
myset1 = {1, 2, 3, 4, 5}
myset2 = {3, 4, 6, 7, 8}

myset3 = myset1 - myset2
print(myset3)
```

аналогичен предыдущему.

Обратите внимание: для операции разности множеств важен порядок, в котором указаны множества. Если поменять местами `myset1` и `myset2`, нас ожидает совсем другой результат: элементы, входящие в множество `myset2` и которых нет в множестве `myset1`.

Приведенный ниже код:

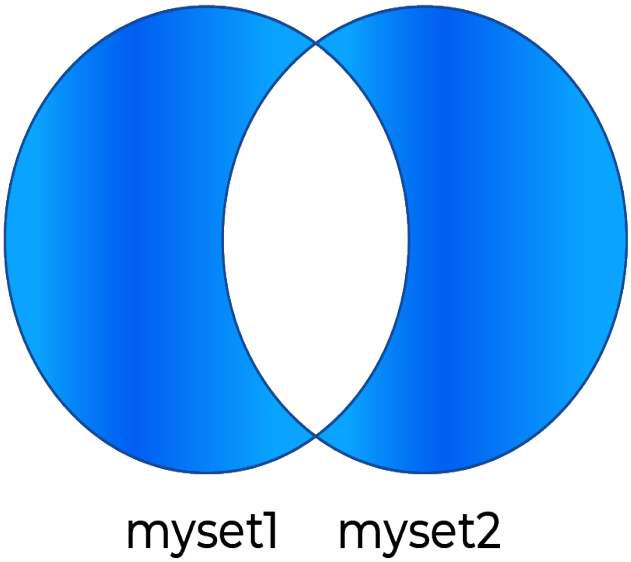
```
myset1 = {1, 2, 3, 4, 5}
myset2 = {3, 4, 6, 7, 8}

myset3 = myset2.difference(myset1)
print(myset3)
```

выводит (порядок элементов может отличаться):

```
{8, 6, 7}
```

Симметрическая разность: метод symmetric_difference()

Симметрическая разность множеств – это множество, включающее все элементы исходных множеств, не принадлежащие одновременно обоим исходным множествам. Для этой операции существует метод **`symmetric_difference()`**.



Приведенный ниже код:

```
myset1 = {1, 2, 3, 4, 5}
myset2 = {3, 4, 6, 7, 8}
```

```
myset3 = myset1.symmetric_difference(myset2)
print(myset3)
```

выводит (порядок элементов может отличаться):

```
{1, 2, 5, 6, 7, 8}
```

Для симметрической разности двух множеств можно также использовать оператор `^`.

Результат выполнения приведенного ниже кода:

```
myset1 = {1, 2, 3, 4, 5}
myset2 = {3, 4, 6, 7, 8}

myset3 = myset1 ^ myset2
print(myset3)
```

аналогичен предыдущему.

Обратите внимание: для операции симметрической разности порядок множеств не важен, на то она и симметрическая: `myset1 ^ myset2 == myset2 ^ myset1`.

Методы множеств, изменяющие текущие множества

Методы **`union()`**, **`intersection()`**, **`difference()`**, **`symmetric_difference()`** не изменяют исходные множества, а возвращают новые. Часто на практике нужно изменять исходные множества. Для таких целей используются парные методы **`update()`**, **`intersection_update()`**, **`difference_update()`**, **`symmetric_difference_update()`**.

Метод update()

Метод **`update()`** изменяет исходное множество **по объединению**.

Приведенный ниже код:

```
myset1 = {1, 2, 3, 4, 5}
myset2 = {3, 4, 6, 7, 8}

myset1.update(myset2)  # изменяем множество myset1
print(myset1)
```

выводит (порядок элементов может отличаться):

```
{1, 2, 3, 4, 5, 6, 7, 8}
```

Аналогичный результат получается, если использовать оператор `|=`:

```
myset1 = {1, 2, 3, 4, 5}
myset2 = {3, 4, 6, 7, 8}

myset1 |= myset2
print(myset1)
```

Метод intersection_update()

Метод **`intersection_update()`** изменяет исходное множество **по пересечению**.

Приведенный ниже код:

```
myset1 = {1, 2, 3, 4, 5}
myset2 = {3, 4, 6, 7, 8}

myset1.intersection_update(myset2)  # изменяем множество myset1
print(myset1)
```

выводит (порядок элементов может отличаться):

```
{3, 4}
```

Аналогичный результат получается, если использовать оператор &=:

```
myset1 = {1, 2, 3, 4, 5}
myset2 = {3, 4, 6, 7, 8}

myset1 &= myset2
print(myset1)
```

Метод difference_update()

Метод `difference_update()` изменяет исходное множество **по разности**.

Приведенный ниже код:

```
myset1 = {1, 2, 3, 4, 5}
myset2 = {3, 4, 6, 7, 8}

myset1.difference_update(myset2)  # изменяем множество myset1
print(myset1)
```

выводит (порядок элементов может отличаться):

```
{1, 2, 5}
```

Аналогичный результат получается, если использовать оператор `-=`:

```
myset1 = {1, 2, 3, 4, 5}
myset2 = {3, 4, 6, 7, 8}

myset1 -= myset2
print(myset1)
```

Метод symmetric_difference_update()

Метод **`symmetric_difference_update()`** изменяет исходное множество **по симметрической разности**.

Приведенный ниже код:

```
myset1 = {1, 2, 3, 4, 5}
```

```
myset2 = {3, 4, 6, 7, 8}

myset1.symmetric_difference_update(myset2)  # изменяем множество myset1
print(myset1)
```

выводит (порядок элементов может отличаться):

```
{1, 2, 5, 6, 7, 8}
```

Аналогичный результат получается, если использовать оператор ^=:

```
myset1 = {1, 2, 3, 4, 5}
myset2 = {3, 4, 6, 7, 8}

myset1 ^= myset2
print(myset1)
```

Примечания

Примечание 1. Все основные операции над множествами выполняются двумя способами: при помощи метода или соответствующего ему оператора. Различие заключается в том, что метод может принимать в качестве аргумента не только множество (тип данных `set`), но и любой итерируемый объект (список, строку, кортеж..).

Приведенный ниже код:

```
mylist = [2021, 2020, 2019, 2018, 2017, 2016]
mytuple = (2021, 2020, 2016)
mystr = 'abcd'

myset = {2009, 2010, 2016}

print(myset.union(mystr))                 # объединяем со строкой
print(myset.intersection(mylist))         # пересекаем со списком
print(myset.difference(mytuple))          # находим разность с кортежем
```

выводит (порядок элементов может отличаться):

```
{2016, 'c', 'b', 'a', 'd', 2009, 2010}
{2016}
{2009, 2010}
```

Приведенный ниже код:

```
mylist = [2021, 2020, 2019, 2018, 2017, 2016]
mytuple = (2021, 2020, 2016)
mystr = 'abcd'

myset = {2009, 2010, 2016}

print(myset | mystr)
print(myset & mylist)
print(myset - mytuple)
```

приводит к возникновению ошибок:

```
TypeError: unsupported operand type(s) for |: 'set' and 'str'
TypeError: unsupported operand type(s) for &: 'set' and 'list'
TypeError: unsupported operand type(s) for -: 'set' and 'tuple'
```

Примечание 2. Некоторые методы (`union()`, `intersection()`, `difference()`) и операторы (`|`, `&`, `-`, `^`) позволяют совершать операции над несколькими множествами сразу.

Приведенный ниже код:

```
myset1 = {1, 2, 3, 4, 5, 6}
myset2 = {2, 3, 4, 5}
myset3 = {5, 6, 7, 8}

union1 = myset1.union(myset2, myset3)
union2 = myset1 | myset2 | myset3

difference1 = myset1.difference(myset2, myset3)
difference2 = myset1 - myset2 - myset3          # порядок выполнения слева-направо

print(union1 == union2)
print(difference1 == difference2)
```

выводит:

```
True
True
```

Примечание 3. Оператор `^` симметрической разности позволяет использовать несколько множеств, а метод `symmetric_difference()` – нет.

Приведенный ниже код:

```
myset1 = {1, 2, 3, 4, 5, 6}
myset2 = {2, 3, 4, 7}
myset3 = {6, 20, 30}

symdifference = myset1 ^ myset2 ^ myset3  # порядок выполнения слева-направо

print(symdifference)
```

выводит (порядок элементов может отличаться):

```
{1, 5, 7, 20, 30}
```

Приведенный ниже код:

```
myset1 = {1, 2, 3, 4, 5, 6}
myset2 = {2, 3, 4, 7}
myset3 = {6, 20, 30}

symdifference = myset1.symmetric_difference(myset2, myset3)
```

```
print(symdifference)
```

приводит к ошибке:

```
TypeError: symmetric_difference() takes exactly one argument (2 given)
```

Примечание 4. Таблица соответствия методов и операторов над множествами.

`A | B` `A.union(B)`	Возвращает множество, являющееся объединением множеств `A` и `B`
`A |= B` `A.update(B)`	Добавляет в множество `A` все элементы из множества `B`
`A & B` `A.intersection(B)`	Возвращает множество, являющееся пересечением множеств `A` и `B`
`A &= B` `A.intersection_update(B)`	Оставляет в множестве `A` только те элементы, которые есть в множестве `B`
`A - B` `A.difference(B)`	Возвращает разность множеств `A` и `B`
`A -= B` `A.difference_update(B)`	Удаляет из множества `A` все элементы, входящие в `B`
`A ^ B` `A.symmetric_difference(B)`	Возвращает симметрическую разность множеств `A` и `B`
`A ^= B` `A.symmetric_difference_update(B)`	Записывает в `A` симметрическую разность множеств `A` и `B`

Примечание 5. Приоритет операторов в порядке убывания (верхние операторы имеют более высокий приоритет, чем нижние) имеет вид:

Оператор	Описание
`-`	разность
`&`	пересечение
`^`	симметрическая разность
`|`	объединение

Тут можно посмотреть про операторы и их приоритеты в Python.

Методы множеств. Часть 3

Подмножества и надмножества

Напомним, что множество `set1` является **подмножеством** множества `set2`, если все элементы первого входят во второе. При этом множество `set2` – **надмножество** множества `set1`.

Любое множество – подмножество самого себя, про такое подмножество говорят "**нестрогое подмножество**".

Метод issubset()

Для определения, является ли одно из множеств подмножеством другого, используется метод **`issubset()`**. Данный метод возвращает значение `True`, если одно множество является подмножеством другого, и `False`, если не является.

Приведенный ниже код:

```
set1 = {2, 3}
set2 = {1, 2, 3, 4, 5, 6}

print(set1.issubset(set2))
```

выводит:

```
True
```

В этом примере `set2` содержит все элементы `set1`. Это означает, что `set1` – подмножество `set2`. Это также означает, что `set2` – надмножество `set1`.

Для определения, является ли одно из множеств подмножеством другого, также применяются операторы **<=** (нестрогое подмножество) и **<** (строгое подмножество).

Приведенный ниже код:

```
set1 = {2, 3}
set2 = {1, 2, 3, 4, 5, 6}

print(set1 <= set2)
```

аналогичен предыдущему.

Метод issuperset()

Для определения, является ли одно из множеств надмножеством другого, используется метод **`issuperset()`**. Данный метод возвращает значение `True`, если одно множество является надмножеством другого, в противном случае он возвращает `False`.

Приведенный ниже код:

```
set1 = {'a', 'b', 'c', 'd', 'e'}
set2 = {'c', 'e'}
```

```
print(set1.issuperset(set2))
```

выводит:

```
True
```

В этом примере `set1` содержит все элементы `set2`. Это означает, что `set1` – надмножество `set2`. Это также означает, что `set2` – подмножество `set1`.

Для определения, является ли одно из множеств надмножеством другого, также применяются операторы `>=` (нестрогое надмножество) и `>` (строгое надмножество).

Приведенный ниже код:

```
set1 = {'a', 'b', 'c', 'd', 'e'}
set2 = {'c', 'e'}

print(set1 >= set2)
```

аналогичен предыдущему.

Метод isdisjoint()

Для определения отсутствия общих элементов в множествах используется метод **`isdisjoint()`**. Данный метод возвращает значение `True`, если множества не имеют общих элементов, и `False`, когда множества имеют общие элементы.

Приведенный ниже код:

```
set1 = {1, 2, 3, 4, 5}
set2 = {5, 6, 7}
set3 = {7, 8, 9}

print(set1.isdisjoint(set2))
print(set1.isdisjoint(set3))
print(set2.isdisjoint(set3))
```

выводит:

```
False
True
False
```

Примечания

Примечание 1. Методы **`issuperset()`**, **`issubset()`**, **`isdisjoint()`** могут принимать в качестве аргумента не только множество (тип данных `set`), но и любой итерируемый объект (список, строку, кортеж...). Сами же эти методы могут применяться только ко множеству (тип данных `set`) или замороженному множеству (тип данных `frozenset`).

Примечание 2. Операторы >, <, >=, <= требуют наличия в качестве операндов множества.

Примечание 3. Таблица соответствия методов и операторов над множествами.

`set1 <= set2` `set1.issubset(set2)`	Возвращает `True`, если `set1` является подмножеством `set2`
`set1 >= set2` `set1.issuperset(set2)`	Возвращает `True`, если `set1` является надмножеством `set2`
`set1 < set2`	Эквивалентно `set1 <= set2 and set1 != set2` (строгое подмножество)
`set1 > set2`	Эквивалентно `set1 >= set2 and set1 != set2` (строгое надмножество)

Генераторы множеств и frozenset

Генераторы множеств

Пусть требуется создать множество, содержащее цифры введенного числа.

Следующий программный код:

```
digits = set(int(input()))
```

приводит к ошибке

```
TypeError: 'int' object is not iterable
```

поскольку функция `set()` принимает в качестве аргумента итерируемый объект, а число (тип данных `int`) таковым не является.

Следующий программный код:

```
digits = set(input())
```

при вводе строки `'12345'` создает множество символов

`{'1', '2', '3', '4', '5'}`, а не множество цифр `{1, 2, 3, 4, 5}`.

Для создания требуемого множества, содержащего уникальные цифры введенного числа, нам придется написать код:

```
digits = set()

for c in input():
    digits.add(int(c))
```

Такой код хоть и не сложен, однако достаточно громоздок.

Для создания множеств в Python можно использовать специальный синтаксис, как при создании списка.

Приведенный выше код можно переписать с использованием генератора множеств:

```
digits = {int(c) for c in input()}
```

Общий вид генератора множеств следующий:

`{выражение for переменная in последовательность}`,

где `выражение` — некоторое выражение, как правило, зависящее от использованной в списочном выражении переменной, которым будут заполнены элементы множества `переменная` — имя некоторой переменной, `последовательность` — последовательность значений, которые она принимает (любой итерируемый объект).

Примеры использования генератора множеств

1. Создать множество, заполненное квадратами целых чисел от 0 до 9, можно так:

```
squares = {i ** 2 for i in range(10)}
```

2. Создать множество, заполненное кубами целых чисел от 10 до 20, можно так:

```
cubes = {i ** 3 for i in range(10, 21)}
```

3. Создать множество, заполненное символами строки, можно так:

```
chars = {c for c in 'abcdefg'}
```

Условия в генераторе множеств

В генераторах множеств можно использовать условный оператор. Например, если требуется создать множество, заполненное только цифрами некоторой строки, то мы можем написать такой код:

```
digits = {int(d) for d in 'abcd12ef78ghj90' if d.isdigit()}
```

Frozenset

Замороженное множество (**`frozenset`**) также является встроенной коллекцией в Python. Обладая характеристиками обычного множества, замороженное множество не может быть изменено после создания.

Кортеж (тип `tuple`) – неизменяемая версия списка (тип `list`), а замороженное множество (тип `frozenset`) – неизменяемая версия обычного множества (тип `set`).

Для создания замороженного множества используется встроенная функция **`frozenset()`**, которая принимает в качестве аргумента другую коллекцию.

Приведенный ниже код:

```
myset1 = frozenset({1, 2, 3})                  # на основе множества
myset2 = frozenset([1, 1, 2, 3, 4, 4, 4, 5, 6, 6])   # на основе списка
myset3 = frozenset('aabcccddee')                 # на основе строки

print(myset1)
print(myset2)
print(myset3)
```

выводит:

```
frozenset({1, 2, 3})
frozenset({1, 2, 3, 4, 5, 6})
frozenset({'e', 'd', 'c', 'b', 'a'})
```

Операции над замороженными множествами

Над замороженными множествами можно производить все операции, которые можно производить над обычными множествами:

- объединение множеств: метод **`union()`** или оператор **`|`**;
- пересечение множеств: метод **`intersection()`** или оператор **`&`**;

- разность множеств: метод **`difference()`** или оператор **`-`**;
- симметрическая разность множеств: метод **`symmetric_difference()`** или оператор **`^`**.

Приведенный ниже код:

```
myset1 = frozenset('hello')
myset2 = frozenset('world')

print(myset1 | myset2)
print(myset1 & myset2)
print(myset1 ^ myset2)
```

выводит:

```
frozenset({'l', 'w', 'e', 'h', 'r', 'd', 'o'})
frozenset({'l', 'o'})
frozenset({'d', 'h', 'w', 'e', 'r'})
```

Результатом операций над замороженными множествами будут тоже замороженные множества.

Примечания

Примечание 1. Будучи изменяемыми, обычные множества не могут быть элементами других множеств. Замороженные множества являются неизменяемыми, а значит могут быть элементами других множеств.

Приведенный ниже код:

```
sentence = 'The cat in the hat had two sidekicks, thing one and thing two.'

words = sentence.lower().replace('.', '').replace(',', '').split()

vowels = ['a', 'e', 'i', 'o', 'u']

consonants = {frozenset({letter for letter in word if letter not in vowels}) for word in words}

print(*consonants, sep='\n')
```

выводит (порядок элементов может отличаться):

```
frozenset({'d', 'h'})
frozenset({'h', 't'})
frozenset({'n', 'h', 'g', 't'})
frozenset({'n'})
frozenset({'c', 't'})
frozenset({'n', 'd'})
frozenset({'w', 't'})
frozenset({'s', 'c', 'k', 'd'})
```

Примечание 2. Методы, изменяющие множество, отсутствуют у замороженных множеств:

- `add()`

- `remove()`
- `discard()`
- `pop()`
- `clear()`
- `update()`
- `intersection_update()`
- `difference_update()`
- `symmetric_difference_update()`

Примечание 3. Мы можем сравнивать простые (тип `set`) и замороженные множества (тип `frozenset`).

Приведенный ниже код:

```
myset1 = set('qwerty')
myset2 = frozenset('qwerty')

print(myset1 == myset2)
```

выведет:

```
True
```