**Множества (set)**

**Множество** – некоторый набор объектов. Множество неупорядочено. В множестве каждый элемент хранится один раз.

Элементами множества могут быть любые неизменяемые объекты int, float, str, bool, tuple (картеж). В множество не могут входить списки.

S={"one", "tow", 1, 2, True, (1,2), (1,3)}

Конфликта с синтаксисом записи словарей не происходит, так как в словаре пары ключ-значение разделяются двоеточием, а во множестве пар нет и все элементы записаны через запятую. Но пустые фигурные скобки уже заняты для создания пустого словаря, поэтому для создания пустых множеств обязательно вызывать функцию set

**Пустое множество**

s = set()

**Создание множества**

а) из букв строки

s = set("Hello")

б) из списка

s = set([1,2,3])

Порядок элементов в множестве будет не известен.

У множеств есть три коренных отличия от списков:

* Порядок элементов во множестве не определён.
* Множество не может содержать одинаковых элементов.
* Элементы множеств — всегда данные неизменяемого типа.

Первые два свойства являются следствиями математического определения множества:

*Множество считается заданным, если для любого объекта можно сказать, принадлежит он множеству или нет.* То есть множество задаётся некоторым высказыванием, которое может быть истинно (элемент принадлежит множеству) или ложно (не принадлежит).

Отметим для себя два момента:

* Множество не упорядочено. Мы можем узнать, принадлежит элемент множеству или нет. Порядок следования элементов узнать нельзя.
* Нельзя сказать, что элемент входит в множество несколько раз. У нас есть только возможность проверить, есть он во множестве или нет. Поэтому будем считать, что если элемент есть во множестве, то он там только один.

Неизменяемость является дополнительным техническим требованием, позволяющим реализовать быстрые операции с множествами. Дело в том, что элементы множества специальным сложным образом упорядочены и при изменении элемента после добавления во множество его пришлось бы переупорядочивать по-новому, что привело бы к дополнительным временным затратам. Поэтому такие вещи запрещены и элементами множеств могут быть только данные неизменяемых типов.

**Операции c множествами**

**1. Перебор элементов множества**

**for** el **in** S:

 print(el,end=" ")

Порядок, в котором элементы будут перебираться может быть произвольным.

e H l o

Если их надо перебирать в упорядоченном виде, то можно множество превратить в список, а список отсортировать

**for** el **in** sorted(list(S)):

 print(el,end=" ")

Код для работы с множествами нужно писать таким образом, чтобы он правильно работал при любом порядке обхода. Для этого надо знать два правила:

* Если мы не изменяли множество, то порядок обхода элементов тоже не изменится.
* После изменения множества порядок элементов может поменяться коренным образом.

**2. Проверка принадлежности**

**if** 5 **in** s:

непринадлежности

**if** **not**(5 **in** s):

**if** 5 **not** **in** s:

**3. Добавление элемента**

s.**add**(x)

Если элемент, равный x, уже существует во множестве, то оно не изменится, поскольку не может содержать одинаковых элементов. Ошибки при этом не произойдёт.

**4.** **Количество элементов в множестве**

**len**(s)

**5. Удаление элемента из множества**

s.**discard**(x)

Если удаляете элемент, которого нет, то ошибки не происходит

s.**remove**(x)

Если удаляете элемент, которого нет, то произойдет ошибка.

На первый взгляд, странно, что есть метод remove, который увеличивает количество падений вашей программы. Однако, если вы на 100 процентов уверены, что элемент должен быть в множестве, то лучше получить ошибку во время отладки и исправить её, чем тратить время на поиски при неправильной работе программы.

y = s.pop()

Метод pop удаляет случайный элемент из множества и возвращает его как результат. Если попытаться применить pop к пустому множеству, произойдёт ошибка KeyError.

**6. Удаление всех элементов из множества**

S.clear()

**7. Объединение множеств**

C = A | B

C=A.**union**(B)

A |= B

**8. Пересечение множеств**

C = A & B

C=A.**intersection**(B)

A &= B

A.**intersection\_update**(B)

**9. Разность множеств**

C = A - B

C=A.**difference**(B)

A -= B

A.difference\_update(B)

**10. Симметричная разность**

C = A ^ B

C=A.symmetric\_difference(B)

A ^= B

A.**symmetric\_difference\_update**(B)

**11. Сравнение**

A == B Оба множества состоят из одинаковых элементов

A != B

A < B

A <= B

Множества считаются равными, когда они содержат одинаковые наборы элементов.

Обратите внимание, что у двух равных множеств могут быть разные порядки обхода, например, из-за того, что элементы в каждое из них добавлялись в разном порядке.

**Преобразование списка в множество**

Функция set преобразует всё, что можно обойти с помощью цикла for, в множество:

При этом повторяющиеся значения будут входить в множество только один раз.

s = set([1, 2, 3, 1, 2]) # s = {1, 2, 3} порядок может быть другим

s = set(' abracadabra') # s = { 'a ', 'b ', 'r ', 'c ', 'd '}

Множество можно преобразовать в список с помощью функции list. Порядок преобразования случайный.

Реализация множества через списки неэффективна по времени. В этом случае определение принадлежности будет выполняться за O(n). Множества на языке Python реализованы более хитрым способом с помощью хеширования и все операции выполняются быстро вне зависимости от размера множества. Это требует отказа от упорядоченности и от отказа хранить в множестве изменяемые элементы.