**Множества в Python**

Множество содержит неупорядоченные уникальные значения. Одно множество может содержать значения любых типов. Если у вас есть два множества, вы можете совершать над ними любые стандартные операции, например, объединение, пересечение и разность.

**Создание множества**

**>>> a\_set = {1} (1)**

**>>> a\_set**

**{1}**

**>>> type(a\_set) (2)**

**<class 'set'>**

**>>> a\_set = {1, 2} (3)**

**>>> a\_set**

**{1, 2}**

Чтобы создать множество с одним значением, поместите его в фигурные скобки ({}).

Множества, вообще-то, реализуются как классы.

Чтобы создать множество с несколькими значениями, отделите их друг от друга запятыми и поместите внутрь фигурных скобок.

Также вы можете создать множество из списка.

**>>> a\_list = ['a', 'b', 'mpilgrim', True, False, 42]**

**>>> a\_set = set(a\_list) (1)**

**>>> a\_set (2)**

**{'a', False, 'b', True, 'mpilgrim', 42}**

Если вы добавляете элементы в множество, оно не запоминает, в каком порядке они добавлялись.

Можно создать пустое множество.

**>>> a\_set = set() (1)**

**>>> a\_set (2)**

**set()**

**>>> type(a\_set) (3)**

**<class 'set'>**

**>>> len(a\_set) (4)**

**0**

**>>> not\_sure = {} (5)**

**>>> type(not\_sure)**

**<class 'dict'>**

1. Чтобы создать пустое множество, вызовите set() без аргументов.
2. Напечатанное представление пустого множества выглядит немного странно. Вы, наверное, ожидали увидеть {}? Это означало бы пустой словарь, а не пустое множество.
3. Несмотря на странное печатное представление, это действительно множество…
4. …и это множество не содержит ни одного элемента.
5. Нельзя создать пустое множество с помощью двух фигурных скобок. На самом деле, они создают пустой словарь, а не множество.

**Изменение множества**

Есть два способа добавить элементы в существующее множество: метод add() и метод update().

**>>> a\_set = {1, 2}**

**>>> a\_set.add(4) (1)**

**>>> a\_set**

**{1, 2, 4}**

**>>> len(a\_set) (2)**

**3**

**>>> a\_set.add(1) (3)**

**>>> a\_set**

**{1, 2, 4}**

**>>> len(a\_set) (4)**

**3**

1. Метод add() принимает один аргумент, который может быть любого типа, и добавляет данное значение в множество.
2. Теперь множество содержит 3 элемента.
3. Множества — содержит уникальные значения. Если попытаться добавить значение, которое уже присутствует в множестве, ничего не произойдет. Это не приведет к возникновению ошибки; просто нулевое действие.
4. Это множество все ещё состоит из 3 элементов.

**>>> a\_set = {1, 2, 3}**

**>>> a\_set**

**{1, 2, 3}**

**>>> a\_set.update({2, 4, 6}) (1)**

**>>> a\_set (2)**

**{1, 2, 3, 4, 6}**

**>>> a\_set.update({3, 6, 9}, {1, 2, 3, 5, 8, 13}) (3)**

**>>> a\_set**

**{1, 2, 3, 4, 5, 6, 8, 9, 13}**

**>>> a\_set.update([10, 20, 30]) (4)**

**>>> a\_set**

**{1, 2, 3, 4, 5, 6, 8, 9, 10, 13, 20, 30}**

1. Метод update() принимает один аргумент — множество, и добавляет все его элементы к исходному множеству. Так, как если бы вы вызывали метод add() и по очереди передавали ему все элементы множества.
2. Повторяющиеся значения игнорируются, поскольку множество не может содержать дубликаты.
3. Вообще-то, вы можете вызвать метод update() с любым количеством параметров. Когда он вызывается с двумя множествами, метод update() добавляет все элементы обоих множеств в исходное множество (пропуская повторяющиеся).
4. Метод update() может принимать объекты различных типов, включая списки. Когда ему передается список, он добавляет все его элементы в исходное множество.

**Удаление элементов множества**

Существуют три способа удаления отдельных значений из множества. Первые два, discard() и remove(), немного различаются.

**>>> a\_set = {1, 3, 6, 10, 15, 21, 28, 36, 45}**

**>>> a\_set**

**{1, 3, 36, 6, 10, 45, 15, 21, 28}**

**>>> a\_set.discard(10) (1)**

**>>> a\_set**

**{1, 3, 36, 6, 45, 15, 21, 28}**

**>>> a\_set.discard(10) (2)**

**>>> a\_set**

**{1, 3, 36, 6, 45, 15, 21, 28}**

**>>> a\_set.remove(21) (3)**

**>>> a\_set**

**{1, 3, 36, 6, 45, 15, 28}**

**>>> a\_set.remove(21) (4)**

**Traceback (most recent call last):**

**File "<stdin>", line 1, in <module>**

**KeyError: 21**

*Перевод сообщения оболочки:*

**Раскрутка стека (список последних вызовов):**

**Файл "<stdin>", строка 1, <модуль>**

**KeyError: 21**

1. Метод discard() принимает в качестве аргумента одиночное значение и удаляет это значение из множества.
2. Если вы вызвали метод discard() передав ему значение, которого нет в множестве, ничего не произойдет, просто нулевое действие.
3. Метод remove() также принимает в качестве аргумента одиночное значение, и также удаляет его из множества.
4. Вот в чём отличие: если значения нет в множестве, метод remove() породит исключение KeyError.

Подобно спискам, множества имеют метод pop().

**>>> a\_set = {1, 3, 6, 10, 15, 21, 28, 36, 45}**

**>>> a\_set.pop() (1)**

**1**

**>>> a\_set.pop()**

**3**

**>>> a\_set.pop()**

**36**

**>>> a\_set**

**{6, 10, 45, 15, 21, 28}**

**>>> a\_set.clear() (2)**

**>>> a\_set**

**set()**

**>>> a\_set.pop() (3)**

**Traceback (most recent call last):**

**File "<stdin>", line 1, in <module>**

**KeyError: 'pop from an empty set'**

*Перевод сообщения оболочки:*

**Раскрутка стека (список последних вызовов):**

**Файл "<stdin>", строка 1, <модуль>**

**KeyError: 'pop из пустого множества'**

1. Метод pop() удаляет один элемент из множества и возвращает его значение. Однако, поскольку множества неупорядочены, это не «последний» элемент в множестве, поэтому невозможно проконтролировать какое значение было удалено. Удаляется произвольный элемент.
2. Метод clear() удаляет все элементы множества, оставляя вас с пустым множеством. Это эквивалентно записи a\_set = set(), которая создаст новое пустое множество и перезапишет предыдущее значение переменной a\_set.
3. Попытка извлечения (pop) элемента из пустого множества породит исключение KeyError.

**Основные операции с множествами**

Тип set в Python поддерживает несколько основных операций над множествами.

**>>> a\_set = {2, 4, 5, 9, 12, 21, 30, 51, 76, 127, 195}**

**>>> 30 in a\_set (1)**

**True**

**>>> 31 in a\_set**

**False**

**>>> b\_set = {1, 2, 3, 5, 6, 8, 9, 12, 15, 17, 18, 21}**

**>>> a\_set.union(b\_set) (2)**

**{1, 2, 195, 4, 5, 6, 8, 12, 76, 15, 17, 18, 3, 21, 30, 51, 9, 127}**

**>>> a\_set.intersection(b\_set) (3)**

**{9, 2, 12, 5, 21}**

**>>> a\_set.difference(b\_set) (4)**

**{195, 4, 76, 51, 30, 127}**

**>>> a\_set.symmetric\_difference(b\_set) (5)**

**{1, 3, 4, 6, 8, 76, 15, 17, 18, 195, 127, 30, 51}**

1. Чтобы проверить, принадлежит ли значение множеству, используйте оператор in. Он работает так же, как и для списков.
2. Метод union() (объединение) возвращает новое множество, содержащее все элементы каждого из множеств.
3. Метод intersection() (пересечение) возвращает новое множество, содержащее все элементы, которые есть и в первом множестве, и во втором.
4. Метод difference() (разность) возвращает новое множество, содержащее все элементы, которые есть в множестве a\_set, но которых нет в множестве b\_set.
5. Метод symmetric\_difference() (симметрическая разность) возвращает новое множество, которое содержит только уникальные элементы обоих множеств.

Три из этих методов симметричны.

**# продолжение предыдущего примера**

**>>> b\_set.symmetric\_difference(a\_set) (1)**

**{3, 1, 195, 4, 6, 8, 76, 15, 17, 18, 51, 30, 127}**

**>>> b\_set.symmetric\_difference(a\_set) == a\_set.symmetric\_difference(b\_set) (2)**

**True**

**>>> b\_set.union(a\_set) == a\_set.union(b\_set) (3)**

**True**

**>>> b\_set.intersection(a\_set) == a\_set.intersection(b\_set) (4)**

**True**

**>>> b\_set.difference(a\_set) == a\_set.difference(b\_set) (5)**

**False**

1. Симметрическая разность множеств a\_set и b\_set выглядит не так, как симметрическая разность множеств b\_set и a\_set, но вспомните, множества неупорядочены. Любые два множества, все (без исключения) значения которых одинаковы, считаются равными.
2. Именно это здесь и произошло. Глядя на печатное представление этих множеств, созданное оболочкой Python, не обманывайтесь. Значения элементов этих множеств одинаковы, поэтому они равны.
3. Объединение двух множеств также симметрично.
4. Пересечение двух множеств также симметрично.
5. Разность двух множеств несимметрична. По смыслу, данная операция аналогична вычитанию одного числа из другого. Порядок операндов имеет значение.

Наконец, есть ещё несколько вопросов по множествам, которые вы можете задать.

**>>> a\_set = {1, 2, 3}**

**>>> b\_set = {1, 2, 3, 4}**

**>>> a\_set.issubset(b\_set) (1)**

**True**

**>>> b\_set.issuperset(a\_set) (2)**

**True**

**>>> a\_set.add(5) (3)**

**>>> a\_set.issubset(b\_set)**

**False**

**>>> b\_set.issuperset(a\_set)**

**False**

1. Множество a\_set является подмножеством b\_set — все элементы a\_set также являются элементами b\_set.
2. И наоборот, b\_set является надмножеством a\_set, потому что все элементы a\_set также являются элементами b\_set.
3. Поскольку вы добавили элемент в a\_set, но не добавили в b\_set, обе проверки вернут значение False.

**Множества в логическом контексте**

Вы можете использовать множества в логическом контексте, например, в операторе if.

**>>> def is\_it\_true(anything):**

**… if anything:**

**… print("да, это истина")**

**… else:**

**… print("нет, это ложь")**

**…**

**>>> is\_it\_true(set()) (1)**

**нет, это ложь**

**>>> is\_it\_true({'a'}) (2)**

**да, это истина**

**>>> is\_it\_true({False}) (3)**

**да, это истина**

1. В логическом контексте пустое множество — ложь.
2. Любое множество, содержащее хотя бы один элемент — истина.
3. Любое множество, содержащее хотя бы один элемент — истина. Значения элементов не важны.

**Множества**

Множество в языке Питон — это структура данных, эквивалентная множествам в математике. Множество может состоять из различных элементов, порядок элементов в множестве неопределен. В множество можно добавлять и удалять элементы, можно перебирать элементы множества, можно выполнять операции над множествами (объединение, пересечение, разность). Можно проверять принадлежность элементу множества.

В отличии от массивов, где элементы хранятся в виде последовательного списка, в множествах порядок хранения элементов не определен (более того, элементы множества хранятся не подряд, как в списке, а при помощи хитрых алгоритмов). Это позволяет выполнять операции типа “проверить принадлежность элемента множеству” быстрее, чем просто перебирая все элементы множества.

Элементами множества может быть любой неизменяемый тип данных: числа, строки, кортежи. Изменяемые типы данных не могут быть элементами множества, в частности, нельзя сделать элементом множества список (но можно сделать кортеж) или другое множество. Требование неизменяемости элементов множества накладывается особенностями представления множества в памяти компьютера.

**Задание множеств**

Множество задается перечислением всех его элементов в фигурных скобках. Например:

**A = {1, 2, 3}**

Исключением является пустое множество, которое можно создать при помощи функции **set()**. Если функции **set** передать в качестве параметра список, строку или кортеж, то она вернет множество, составленное из элементов списка, строки, кортежа. Например:

**A = set('qwerty')**

**print(A)**

выведет **{’e’, ‘q’, ‘r’, ‘t’, ‘w’, ’y’}**.

Каждый элемент может входить в множество только один раз, порядок задания элементов не важен. Например, программа:

**A = {1, 2, 3}**

**B = {3, 2, 3, 1}**

**print(A == B)**

выведет **True**, так как A и B — равные множества.

Каждый элемент может входить в множество только один раз. **set(‘Hello’)** вернет множество из четырех элементов: **{’H’, ‘e’, ‘l’, ’o’}**.

**Работа с элементами множеств**

Узнать число элементов в множестве можно при помощи функции **len**, как в случае с массивами и строками. Например, **len(C)**.

Проверить, принадлежит ли элемент множеству можно при помощи операции **in**, возвращающей значение типа **bool**, то есть **True** (истина) или **False** (ложь).

**i in A**

Аналогично есть противоположная операция **not in**.  
Для добавления элемента в множество есть метод **add**:

**A.add(x)**

Для удаления элемента x из множества есть два метода: **discard** и **remove**. Их поведение различается только в случае, когда удаляемый элемент отсутствует в множестве. В этом случае метод **discard** не делает ничего, а метод **remove** генерирует исключение **KeyError**.

Наконец, метод **pop** удаляет из множества один случайный элемент и возвращает его значение. Если же множество пусто, то генерируется исключение **KeyError**.

Из множества можно сделать список при помощи функции **list**.

**Перебор элементов множества**

При помощи цикла **for** можно перебрать все элементы множества:

**Primes = {2, 3, 5, 7, 11}**

**for num im Primes:**

**print(num)**

**Операции с множествами**

С множествами в питоне можно выполнять обычные для математики операции над множествами.

|  |  |
| --- | --- |
| Операция | Описание |
| 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 (элементы, входящие в A, но не входящие в B) |
| A -= B  A.difference\_update(B) | Удаляет из множества A все элементы, входящие в множество B |
| A ^ B  A.symmetric\_difference(B) | Возвращает симметрическую разность множеств A и B (элементы, входящие в A или в B, но не в оба из них одновременно) |
| A ^= B  A.symmetric\_difference\_update(B) | Записывает в A симметрическую разность множеств A и B |
| A <= B  A.issubset(B) | Возвращает True, естли A является подмножеством B |
| A >= B  A.issuperset(B) | Возвращает True, если A является надмножеством B |
| A < B | Эквивалентно A <= B and A != B |
| A > B | Эквивалентно A >= B and A != B |

При использовании формы A.update(B) и аналогичных, параметр B может также являться списком, строкой и вообще любым итерируемым объектом. Так, запись A.update([1, 2, 1]) - корректна (в множество будут добавлены элементы 1 и 2), а запись A |= [1, 2, 1] - породит исключение.