12. Работа со множествами

- Множества
- Методы множеств
- Операции множеств

set в Python – это встроенный тип, предлагающий широкий набор возможностей, которые повторяют теорию множеств из математики. Тем не менее интерпретация может отличаться от той, что принята в математике.

Множества

Множества – это **неупорядоченная** коллекция **уникальных** элементов, сгруппированных под одним именем. Множество может быть *неоднородным* – включать элементы разных типов. Множество всегда состоит только из *уникальных элементов* (дубли запрещены) в отличие от списков и кортежей.

Чаще всего множества в Python используются для проверки на принадлежность, удаления повторов из последовательности и выполнения математических операций, таких как пересечение, объединение, поиск разностей и симметрических разностей.

Создание множеств

Есть два способа создать set объект в Python:

1. Использовать фигурные скобки {}
2. Использовать встроенную функцию set()

```
>>> var_2 = {1, 'python.org', 57.45}
```

Рисунок.1 – Создание множеств с помощью фигурных скобок

Множество создается при размещении всех элементов внутри фигурных скобок {} как показано на *Рис.1*

Еще один способ создать (или определить) множество в Python — использовать функцию set()

```
>>> var_1 = set()  # Это создаст пустое множество
>>> data_list = [94, 'This string', 45.10]  # список
>>> var_2 = set(data_list)
```

Рисунок.1.1 – Создание множеств с помощью set()

Нет ограничений на количество элементов в объекте set, но ***запрещено добавлять элементы изменяемых типов***, такие как список или словарь. Если попробовать добавить список (с набором элементов), интерпретатор выдаст ошибку *Рис.1.2*

```
>>> var_1 = { 10, 20, 30, [50, 60, 70, 80], 90}
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: unhashable type: 'list'
```

Рисунок.1.2 – Создание множеств

Методы множеств

Добавление элементов

Объекты set в Python поддерживают добавление элементов двумя способами: по одному с помощью метода add() или группами с помощью update()

set.add() – добавляет элемент x в set

Один элемент можно добавить с помощью метода s.add()

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

Рисунок.2 – Добавление одного элемента в множество

set.update() добавляет в set, элементы из одной или более последовательности поддерживающих итерирование

Для добавления нескольких элементов необходимо использовать метод update(), который принимает итерабельный объект (список, кортеж, генератор, множество и т.п.)

```
>>> nums_set = {1, 2, 3}
>>> nums_set.update([4, 5, 6])
>>> nums_set
{1, 2, 3, 4, 5, 6}
```

Рисунок.2.1 – Добавление одного элемента в множество

Удаление элементов

Один или несколько элементов можно удалить из объекта set с помощью следующих методов. Их отличие в виде возвращаемого значения

- remove()
- discard()
- pop()

set.remove() удаляет указанный элемент

Метод remove() полезен в тех случаях, когда нужно удалить из множества конкретный элемент и вернуть ошибку в том случае, если его нет в множестве

```
>>> data_set = {1, 4, 6, 3, 2, 'a', 'b', 'c'}
>>> data_set.remove('b')
>>> data_set
{1, 2, 3, 4, 'c', 6, 'a'}
>>> data_set.remove(19) # KeyError (элемент не найден)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
KeyError: 19
```

Рисунок.3 – Метод remove()

set.discard() удаляет указанный элемент

Метод discard() полезен, поскольку он удаляет определенный элемент и не возвращает ошибку, если элемент не был найден в множестве

```
>>> data_set = {1, 4, 6, 3, 2, 'a', 'b', 'c'}
>>> data_set.discard('c')
>>> data_set
{1, 2, 3, 4, 6, 'a', 'b'}
>>> data_set.discard(10)
>>> data_set
{1, 2, 3, 4, 6, 'a', 'b'}
```

Рисунок.3.1 – Метод discard()

set.pop() удаляет и возвращает последний элемент

Метод pop() удаляет по одному элементу за раз в случайном порядке. set – это *неупорядоченная коллекция*, поэтому pop() не требует аргументов (индексов в этом случае). Метод pop() можно воспринимать как неконтролируемый способ удаления элементов по одному из множеств в Python

```
>>> data_set = {'b', 1, 'a'}
>>> data_set.pop() # случайный элемент будет удален
1
>>> data_set.pop() # случайный элемент будет удален
'a'
>>> data_set.pop() # случайный элемент будет удален
'b'
>>> data_set.pop() # KeyError (удаление из пустого set)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
KeyError: 'pop from an empty set'
```

Рисунок.3.2 – Метод pop()

Часто используемые функции множеств

Вы уже знакомы с add(), update(), remove(), pop() и discard(), вот на какие также стоит обратить внимание

Оператор принадлежности (членства)

in проверяет на наличие конкретного элемента в множестве

```
>>> nums_set = {13, 6, 8, 23, 4.7, 76}
>>> 4.7 in nums_set
True
>>> 10 in nums_set
False
>>> 10 not in nums_set
True
>>> 23 not in nums_set
False
```

Рисунок.4 – Оператор членства

len() – вернет количество элементов в объекте set

```
>>> data_set = {1, 4, 6, 3, 2, 'a', 'b', 'c'}
>>> len(data_set)
8
```

Рисунок.5 – Функция len()

set.copy() возвращает копию set

Метод copy() создает копию существующего множества и сохраняет ее в новом объекте

```
>>> data_set = {1, 4, 6, 3, 2, 'a', 'b', 'c'}
>>> data_set_2 = data_set.copy()
>>> data_set_2
{1, 2, 3, 4, 'c', 6, 'a', 'b'}
>>> data_set
{1, 2, 3, 4, 'c', 6, 'a', 'b'}
```

Рисунок.6 – Метод copy()

set.clear() удаляет элементы из set

Метод clear() очищает множество (удаляет все элементы за раз)

```
>>> data_set = {1, 4, 6, 3, 2, 'a', 'b', 'c'}
>>> data_set.clear()
>>> data_set
set()
```

Рисунок.7 – Метод clear()

del удаляет множество целиком

```
>>> nums_set = {13, 6, 8, 23, 4.7, 76}
>>> del nums_set
>>> nums_set
Traceback (most recent call last):
  File "<input>", line 1, in <module>
NameError: name 'nums_set' is not defined
```

Рисунок.7 – Ключевое слово del

Операции множеств

Объединение множеств

При использовании на двух множествах вы получаете новый объект, содержащий элементы обоих (без повторов).

Операция объединения в Python выполняется двумя способам: с помощью символа | или метода set.union()

```
>>> A = {1, 2, 3}
>>> B = {2, 3, 4, 5}
>>> char = A | B  # используя символьный метод
>>> method = A.union(B) # используя метод union()
>>> char
{1, 2, 3, 4, 5}
>>> method
{1, 2, 3, 4, 5}
```

Рисунок.8 – Объединение множеств

Пересечение множеств

При использовании на двух множествах вы получаете новый объект, содержащий общие элементы обоих (без повторов).

Операция пересечения выполняется двумя способами: с помощью символа & или метода set.intersection()

```
>>> A = {1, 2, 3, 4}
>>> B = {3, 4, 5, 6}
>>> char = A & B  # используя символьный метод
>>> method = A.intersection(B)  # используя метод intersection()
>>> char
{3, 4}
>>> method
{3, 4}
```

Рисунок.9 – Пересечение множеств

Разность множеств

При использовании на двух множествах вы получаете новый объект, содержащий элементы, которые есть в первом, но не втором (в данном случае — в множестве «A»).

Операция разности выполняется двумя способами: с помощью символа - или метода difference()

```
>>> A = {1, 2, 3, 4}
>>> B = {3, 4, 5, 6}
>>> char = A - B  # используя символьный метод
>>> method = A.difference(B)  # используя метод difference()
>>> char
{1, 2}
>>> method
{1, 2}
```

Рисунок.10 – Разность множеств

Симметричная разность множеств

При использовании на двух множествах вы получаете новый объект, содержащий все элементы, кроме тех, что есть в обоих.

Симметрическая разность выполняется двумя способами: с помощью символа ^ или метода symmetric_difference()

```
>>> A = {1, 2, 3, 4}
>>> B = {3, 4, 5, 6}
>>> char = A ^ B  # используя символьный метод
>>> method = A.symmetric_difference(B)  # используя метод symmetric_difference()
>>> char
{1, 2, 5, 6}
>>> method
{1, 2, 5, 6}
```

Рисунок.11 – Симметричная разность множеств

Подмножество и надмножество

Множество **B** называется *подмножество* **A**, если все элементы **B** есть в **A**.

Проверить на подмножество в Python можно двумя способами: с помощью символа <= или метода issubset(). Возвращает True или False в зависимости от результата

```
>>> A = {1, 2, 3, 4, 5}
>>> B = {2, 3, 4}
>>> B <= A  # используя символьный метод
True
>>> B.issubset(A)  # используя метод issubset()
True
```

Рисунок.11 – Подмножество

Множество **A** называется *надмножество* **B**, если все элементы **B** есть в **A**.

Проверить на надмножество в Python можно двумя способами: с помощью символа `>=` или метода `issuperset()`. Он возвращает True или False в зависимости от результата

```
>>> A = {1, 2, 3, 4, 5}
>>> B = {2, 3, 4}
>>> A >= B  # используя символьный метод
True
>>> A.issuperset(B)  # используя метод issuperset()
True
```

Рисунок.11 – Надмножество

«Методы множеств» Таблица.1

№	Метод	Описание
1	add()	Добавляет элемент в множество
2	clear()	Удаляет все элементы из множества
3	copy()	Возвращает копию множества
4	difference()	Возвращает новое множество – разность двух или более множеств
5	difference_update()	Удаляет все элементы одного набора из другого
6	discard()	Удаляет элемент, если он содержится в множестве (если элемента в множестве нет, то ничего не происходит)
7	intersection()	Возвращает новое множество – пересечение двух множеств
8	intersection_update()	Добавляет в множество пересечение с другим множеством или с самим собой
9	isdisjoint()	Возвращает True, если два множества не имеют пересечения
10	issubset()	Возвращает True, если определенное множество содержится в другом множестве
11	issuperset()	Возвращает True, если в множестве есть другое множество
12	pop()	Удаляет и возвращает случайный элемент множества. Если

		множество пусто, то возвращает ошибку KeyError
13	remove()	Удаляет определенный элемент множества. Если элемент отсутствует в множестве, то возвращает ошибку KeyError
14	symmetric_difference()	Возвращает новое множество – симметрическую разность двух множеств
15	symmetric_difference_update()	Добавляет в множество симметрическую разницу с другим множеством или с самим собой
16	union()	Возвращает новое множество – объединение множеств
17	update()	Добавляет в множество объединение с другим множеством или с самим собой