Множества в Python

- Множество в языке Питон – это структура данных, эквивалентная множествам в математик
- Множества в Python – это структура данных, которые содержат неупорядоченные элементы. Элементы также не является индексированным
- Так как элементы не индексируются, множества не поддерживают никаких операций среза и индексирования.

запустить выполнить пошагово

```
1  A = {1, 2, 3}
2  A = set('qwerty')
3  print(A)
4
```

Выходные данные:

```
1  set({'q', 'w', 'e', 'r', 't', 'y'})
2
```

запустить выполнить пошагово

```
1  A = {1, 2, 3}
2  B = {3, 2, 3, 1}
3  print(A == B)
4
```

Выходные данные:

```
1  True
2
```

- 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'}.

Создания множества в Python

создать множество путем передачи всех элементов множества внутри фигурных скобок {} и разделить элементы при помощи запятых (,).

Множество может содержать любое количество элементов и элементы могут быть разных типов, к примеру, целые числа, строки, кортежи, и т. д.

Однако, множество не поддерживает изменяемые элементы, такие как списки, словари, и так далее.

Создания множества в Python

создать множество путем передачи всех элементов множества внутри фигурных скобок {} и разделить элементы при помощи запятых (,).

Множество может содержать любое количество элементов и элементы могут быть разных типов, к примеру, целые числа, строки, кортежи, и т. д.

Однако, множество не поддерживает изменяемые элементы, такие как списки, словари, и так далее.

Рассмотрим пример создания множества в Python:

```python
num_set = {1, 2, 3, 4, 5, 6}
print(num_set)
```

Результат:

```python
{1, 2, 3, 4, 5, 6}
```

```python
string_set = {"Nicholas", "Michelle", "John", "Mercy"}
print(string_set)
```

Результат:

```python
{'Michelle', 'Nicholas', 'John', 'Mercy'}
```

Мы также можем создать множество с элементами разных типов. Например:

```python
mixed_set = {2.0, "Nicholas", (1, 2, 3)}
print(mixed_set)
```

Результат:

```python
{2.0, 'Nicholas', (1, 2, 3)}
```

Мы также можем создать множество из списков. Это можно сделать, вызвав встроенную функцию Python под названием `set()`. Например:

```python
num_set = set([1, 2, 3, 4, 5, 6])
print(num_set)
```

Результат:

```python
{1, 2, 3, 4, 5, 6}
```

```python
num_set = set([1, 2, 3, 1, 2])
print(num_set)
```

Результат:

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

Множество удалило дубликаты и выдало только по одному экземпляру элементов. Это также происходит при **создании множества** с нуля. Например:

```python
num_set = {1, 2, 3, 1, 2}
print(num_set)
```

Создание **пустого множества** подразумевает определенную хитрость. Если вы используете пустые фигурные скобки `{}` в Python, вы скорее создадите **пустой словарь**, а не множество. Например:

```python
x = {}
print(type(x))
```

Результат:

```
<class 'dict'>
```

Как показано в выдаче, тип переменной `x` является словарем.

Чтобы создать пустое **множество в Python**, мы должны использовать функцию `set()` без передачи какого-либо значения в параметрах, как показано ниже:

```python
x = set()
print(type(x))
```

Доступ к элементам множеств

Python не предоставляет прямой способ получения значения к отдельным **элементам множества**. Однако, мы можем использовать цикл для итерации через все элементы множества. Например:

```python
months = set(["Jan", "Feb", "March", "Apr", "May", "June", "July", "Aug",
"Sep", "Oct", "Nov", "Dec"])

for m in months:
    print(m)
```

Результат:

```
March
Feb
Dec
Jan
May
Nov
Oct
Apr
June
Aug
Sep
July
```

Мы также можем проверить наличие элемента во множестве при помощи `in`, как показано ниже:

```python
months = set(["Jan", "Feb", "March", "Apr", "May", "June", "July", "Aug", "Sep", "Oct", "Nov", "Dec"])

print("May" in months)
```

Результат:

```
True
```

Добавление элементов во множество

Python позволяет нам вносить новые элементы во множество при помощи функции `add()`.

Например:

```python
months = set(["Jan", "March", "Apr", "May", "June", "July", "Aug", "Sep", "Oct", "Nov", "Dec"])

months.add("Feb")
print(months)
```

Результат:

```python
{'Oct', 'Dec', 'Feb', 'July', 'May', 'Jan', 'June', 'March', 'Sep', 'Aug', 'Nov', 'Apr'}
```

Удаление элемента из множеств

Python позволяет нам **удалять элемент из множества**, но не используя индекс, так как множество элементов не индексированы. Элементы могут быть удалены при помощи обоих методов `discard()` и `remove()`.

Помните, что метод `discard()` не будет выдавать ошибку, если элемент **не был найден во множестве**. Однако, если метод `remove()` используется и элемент не был найден, возникнет ошибка.

Давайте продемонстрируем как удалять элемент при помощи метода `discard()`:

Python

```python
num_set = {1, 2, 3, 4, 5, 6}
num_set.discard(3)
print(num_set)
```

Результат:

Python

```
{1, 2, 4, 5, 6}
```

С методом `pop()`, мы можем удалить и вернуть элемент. Так как элементы находятся в произвольном порядке, мы не можем утверждать или предсказать, какой элемент будет удален.

Например:

```python
num_set = {1, 2, 3, 4, 5, 6}
print(num_set.pop())
```

Результат:

```python
1
```

Вы можете использовать тот же метод при **удалении элемента** и возврате элементов, которые остаются во множестве. Например:

```python
num_set = {1, 2, 3, 4, 5, 6}
num_set.pop()
print(num_set)
```

Результат:

```python
{2, 3, 4, 5, 6}
```

Объединение множеств

Предположим, у нас есть два множества, **A** и **B**. Объединение этих двух множеств — это множество со всеми элементами обеих множеств. Такая операция выполняется при помощи функции Python под названием `union()`.

Рассмотрим пример:

```python
months_a = set(["Jan", "Feb", "March", "Apr", "May", "June"])
months_b = set(["July", "Aug", "Sep", "Oct", "Nov", "Dec"])

all_months = months_a.union(months_b)
print(all_months)
```

Результат:

```python
{'Oct', 'Jan', 'Nov', 'May', 'Aug', 'Feb', 'Sep', 'March', 'Apr', 'Dec', 'June', 'July'}
```

Если вы хотите **создать объединение** из более двух множеств, разделите названия множеств при помощи оператора `|` . Взглянем на пример:

```python
x = {1, 2, 3}
y = {4, 3, 6}
z = {7, 4, 9}

print(x | y | z)
```

Результат:

```python
{1, 2, 3, 4, 6, 7, 9}
```

Пересечение множеств

Предположим, у вас есть два множества: **A** и **B**. Их пересечение представляет собой множество элементов, которые являются общими для **A** и для **B**.

Операция пересечения во множествах может быть достигнута как при помощи оператора `&`, так и метода `intersection()`. Рассмотрим пример:

Python

```python
x = {1, 2, 3}
y = {4, 3, 6}

print(x & y) # Результат: 3
```

Разница между множествами

Предположим, у вас есть два множества: **A** и **B**. Разница между A и B (A — B) — это множество со всеми элементами, которые содержатся в A, но не в B. Соответственно, (B — A) — это множество со всеми элементами в B, но не в A.

КОД

Для определения разницы между множествами в Python, мы можем использовать как функцию difference(), так и оператор — . Рассмотрим пример:

```python
set_a = {1, 2, 3, 4, 5}
set_b = {4, 5, 6, 7, 8}
diff_set = set_a.difference(set_b)
print(diff_set)
```

Результат:

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

Симметричная разница между множествами A и B — это множество с элементами, которые находятся в **A** и **B**, за исключением тех элементов, которые **являются общими** для обеих множеств. Это определяется использованием метода Python под названием `symmetric_difference()`, или оператора `^`. Посмотрим на пример:

```python
set_a = {1, 2, 3, 4, 5}
set_b = {4, 5, 6, 7, 8}
symm_diff = set_a.symmetric_difference(set_b)
print(symm_diff)
```

Результат:

```python
{1, 2, 3, 6, 7, 8}
```

Симметричную разницу можно также найти следующим образом:

```python
set_a = {1, 2, 3, 4, 5}
set_b = {4, 5, 6, 7, 8}
print(set_a ^ set_b)
```

Результат:

```python
{1, 2, 3, 6, 7, 8}
```

Методы множеств

Python содержит огромное количество встроенных методов, включая следующие:

Метод copy()

Этот метод возвращает копию множества. Например:

```python
string_set = {"Nicholas", "Michelle", "John", "Mercy"}
x = string_set.copy()

print(x)
```

Результат:

```python
{'John', 'Michelle', 'Nicholas', 'Mercy'}
```

Метод isdisjoint()

Этот метод проверяет, является ли множество пересечением или нет. Если множества **не содержат общих элементов**, метод возвращает `True`, в противном случае — `False`. Например:

```python
names_a = {"Nicholas", "Michelle", "John", "Mercy"}
names_b = {"Jeff", "Bosco", "Teddy", "Milly"}

x = names_a.isdisjoint(names_b)
print(x)
```

Результат:

```python
True
```

Оба множества **не имеют общих элементов**, что делает выдачу `True`.

Метод len()

Этот метод возвращает **длину множества**, которая является общим количеством элементов во множестве. Пример:

```python
names_a = {"Nicholas", "Michelle", "John", "Mercy"}

print(len(names_a)) # Результат: 4
```

Выдача показывает, что длина множества является 4.

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 `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`, если `A` является подмножеством `B`.
A >= B **A.issuperset(B)**	Возвращает `true`, если `B` является подмножеством `A`.
A < B	Эквивалентно `A <= B and A != B`
A > B	Эквивалентно `A >= B and A != B`