Множества (Set)

Что такое множество?

Множество в Python представляет собой неупорядоченную коллекцию уникальных элементов. Это означает, что каждый элемент в множестве уникален, то есть в множестве не может быть двух одинаковых элементов.

Основные свойства множеств:

- *Уникальность* элементов: Каждый элемент в множестве уникален, дубликаты не допускаются. Если вы попытаетесь добавить в множество элемент, который уже существует в нем, то он просто будет проигнорирован.
- *Неупорядоченность*: Элементы в множестве не упорядочены и не имеют индексов. Это означает, что порядок элементов в множестве не гарантирован и может изменяться при каждой операции.
- *Изменяемость*: Множества в Python изменяемы, то есть вы можете изменять их, добавлять и удалять элементы после создания.

Создание множеств в Python может быть выполнено несколькими способами, включая использование фигурных скобок {} и функции set().

Использование фигурных скобок {}:

```
my_set = {1, 2, 3, 4, 5}  # создание множества с элементами
```

Этот способ подходит, когда вы уже знаете все элементы, которые должны находиться в множестве.

Использование функции set():

```
my_list = [1, 2, 3, 4, 5]
my_set = set(my_list)  # создание множества из списка
```

Также можно создать множество с помощью генератора множеств:

```
my_set = {x for x in range(1, 6)}
```

Этот способ полезен, когда вы хотите создать множество на основе некоторого выражения или итерируемого объекта.

При создании множества Python автоматически удаляет дубликаты, если они есть в исходном объекте. Например:

```
my_list = [1, 2, 3, 4, 4, 5, 5]
my_set = set(my_list)
print(my_set)  # вывод: {1, 2, 3, 4, 5} - дубликаты удалены
```

Методы set

Метод add() используется для добавления нового элемента в множество.

```
my_set = {1, 2, 3}
my_set.add(4)
print(my_set)  # вывод: {1, 2, 3, 4}
```

Метод remove() удаляет указанный элемент из множества. Если элемент отсутствует, он вызывает исключение KeyError.

```
my_set = {1, 2, 3}
my_set.remove(2)
print(my_set)  # вывод: {1, 3}
```

Метод discard() также удаляет указанный элемент из множества. Однако, если элемент отсутствует, метод не вызывает исключение.

```
my_set = {1, 2, 3}
my_set.discard(2)
print(my_set)  # вывод: {1, 3}
```

Метод pop() удаляет и возвращает случайный элемент из множества. Если множество пусто, метод вызывает исключение KeyError.

```
my_set = {1, 2, 3}
popped_element = my_set.pop()
print(popped_element)  # вывод: случайный элемент
```

Метод update для множеств (set) в Python используется для обновления множества, добавляя в него элементы из других итерируемых объектов, таких как другие множества, списки или кортежи.

```
# Исходное множество
my_set = {1, 2, 3}

# Другое множество для добавления
other_set = {4, 5, 6}

# Добавляем элементы из другого множества
my_set.update(other_set)

print(my_set)  # Вывод: {1, 2, 3, 4, 5, 6}
```

Метод clear() удаляет все элементы из множества, оставляя его пустым.

Метод union() для объединения множеств: метод union() возвращает новое множество, содержащее все уникальные элементы из обоих множеств.

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

# Метод union()
union_set_method = set1.union(set2)
print(union_set_method)  # вывод: {1, 2, 3, 4, 5}

# Оператор |
union_set_operator = set1 | set2
print(union_set_operator)  # вывод: {1, 2, 3, 4, 5}
```

Метод intersection() для нахождения пересечения множеств: метод intersection() возвращает новое множество, содержащее элементы, которые присутствуют в обоих множествах.

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

# Метод intersection()
intersection_set_method = set1.intersection(set2)
print(intersection_set_method)  # вывод: {3}

# Оператор &
intersection_set_operator = set1 & set2
print(intersection_set_operator)  # вывод: {3}
```

Метод difference() для нахождения разности множеств: метод difference() возвращает новое множество, содержащее элементы, присутствующие только в одном из множеств.

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

# Метод difference()
difference_set_method = set1.difference(set2)
print(difference_set_method)  # вывод: {1, 2}

# Оператор -
difference_set_operator = set1 - set2
print(difference_set_operator)  # вывод: {1, 2}
```

Метод symmetric_difference() возвращает новое множество, содержащее элементы, присутствующие только в одном из множеств, но не в обоих.

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
symmetric_difference_set = set1.symmetric_difference(set2)
print(symmetric_difference_set)  # вывод: {1, 2, 4, 5}
```

Метод copy(): Создает копию множества.

```
set1 = {1, 2, 3}
set2 = set1.copy()
print(set2)  # вывод: {1, 2, 3}
```

Практические задания

1. Уникальные элементы в списке: Напишите функцию unique_elements, которая принимает список в качестве аргумента и возвращает множество уникальных элементов этого списка.
2. Пересечение двух списков: Напишите функцию common_elements, которая принимает два списка в качестве аргументов и возвращает множество общих элементов этих списков.
3. Разность двух множеств: Напишите функцию set_difference, которая принимает два множества в качестве аргументов и возвращает множество элементов, которые есть в первом множестве, но отсутствуют во втором.
4. Объединение списка и множества: Напишите функцию union_list_set, которая принимает список и множество в качестве аргументов и возвращает множество, содержащее все уникальные элементы из списка и множества.
5. Удаление из списка элементов, содержащихся в множестве: Напишите функцию remove_items_in_set, которая принимает список и множество в качестве аргументов и возвращает новый список, не содержащий элементы, которые присутствуют в множестве.
6. Симметрическая разность списков: Напишите функцию symmetric_difference_lists, которая принимает два списка в качестве аргументов и возвращает список, содержащий элементы, присутствующие только в одном из списков.
7. Получение элементов из списка, не присутствующих в множестве: Напишите функцию get_items_not_in_set, которая принимает список и множество в качестве аргументов и возвращает множество элементов из списка, которые не содержатся в множестве.

8. Поиск общих элементов в нескольких списках: Напишите функцию common_elements_multiple_lists, которая принимает произвольное количество списков в качестве аргументов и возвращает множество, содержащее все общие элементы из всех переданных списков.