Множества (set)

Множества (set) в Python — это неупорядоченная коллекция уникальных элементов. Они были добавлены в язык в версии 2.4 и являются реализацией математической концепции множества.

Основные свойства множеств в Python:

1. Неупорядоченность — элементы множества не имеют определенного порядка, т.е. множество {1, 2, 3} эквивалентно множеству {3, 2, 1}.
2. Уникальность — все элементы множества должны быть уникальными, т.е. множество {1, 2, 2, 3} эквивалентно множеству {1, 2, 3}.
3. Мутабельность — множества могут быть изменены после создания, т.е. к ним можно добавлять или удалять элементы.
4. Хешируемость — элементы множества должны быть хешируемыми, т.е. должны иметь хеш-значение, которое используется для быстрого поиска элемента в множестве.

Создание множеств в Python:

1. С помощью фигурных скобок {}:

```
my_set = {1, 2, 3}
```

2. С помощью функции set():

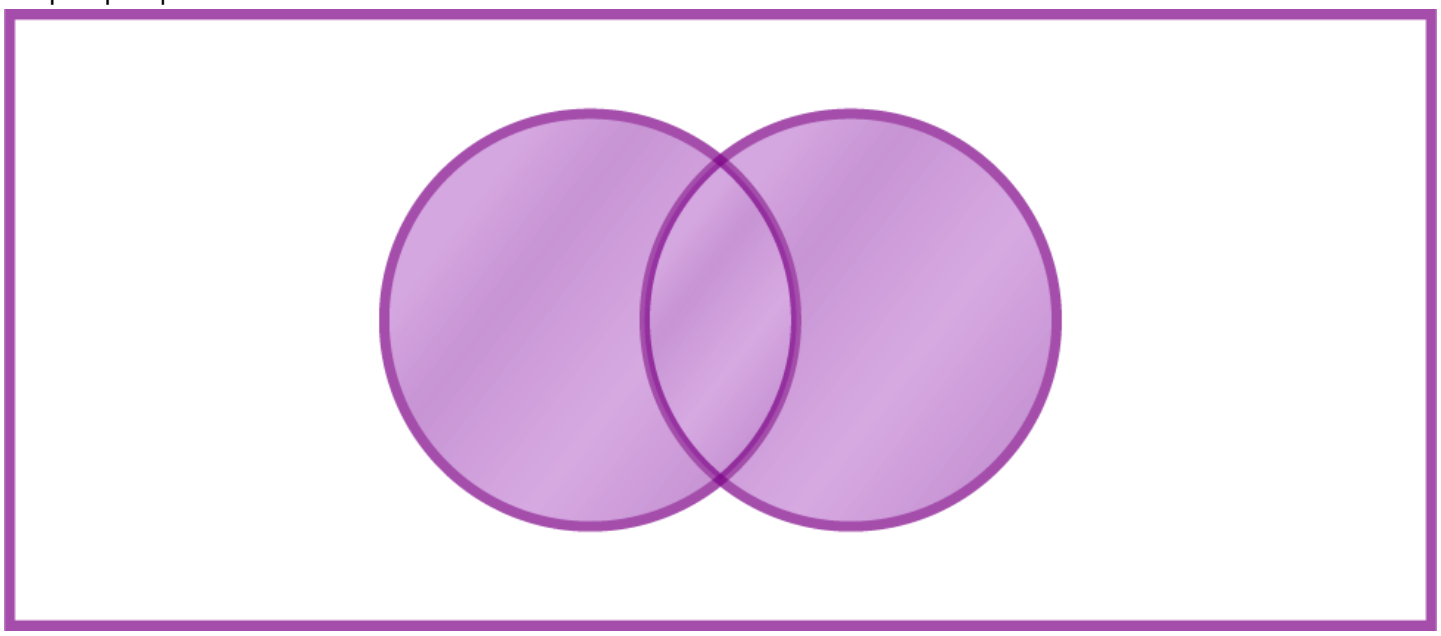
```
my_set = set([1, 2, 3])
```

3. С помощью генератора множеств:

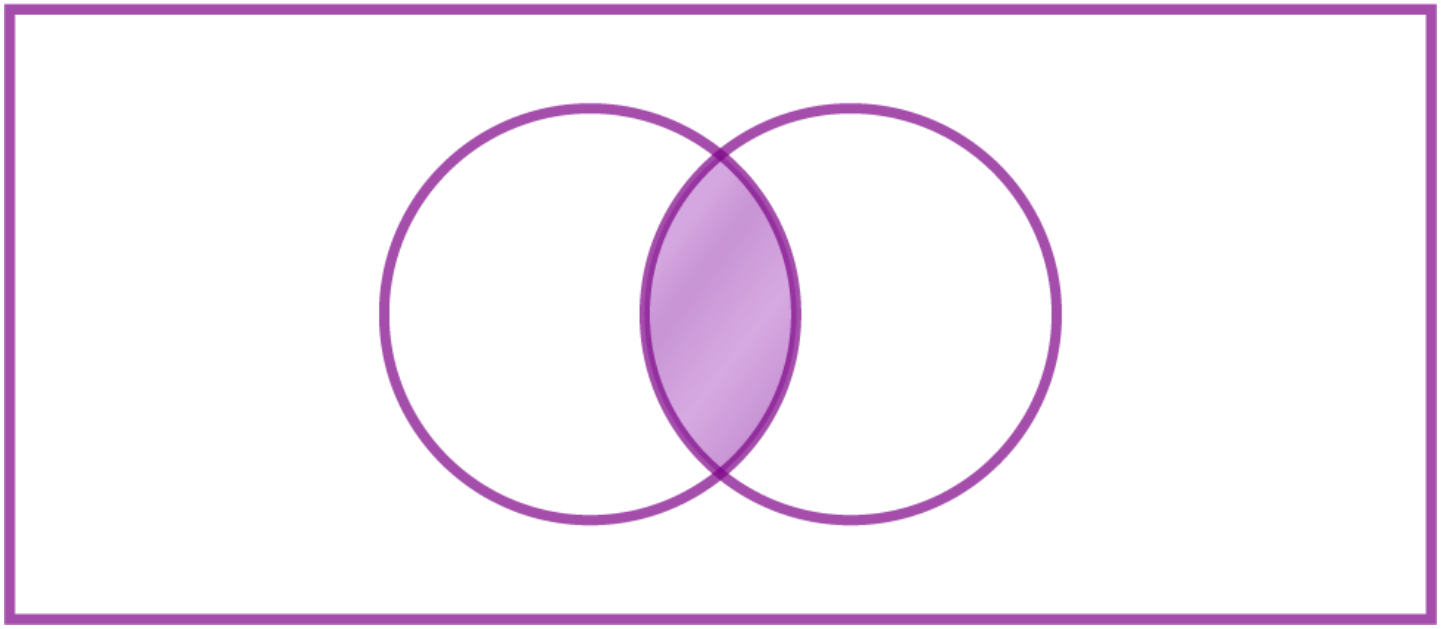
```
my_set = {x for x in range(1, 4)}
```

Основные операции с множествами:

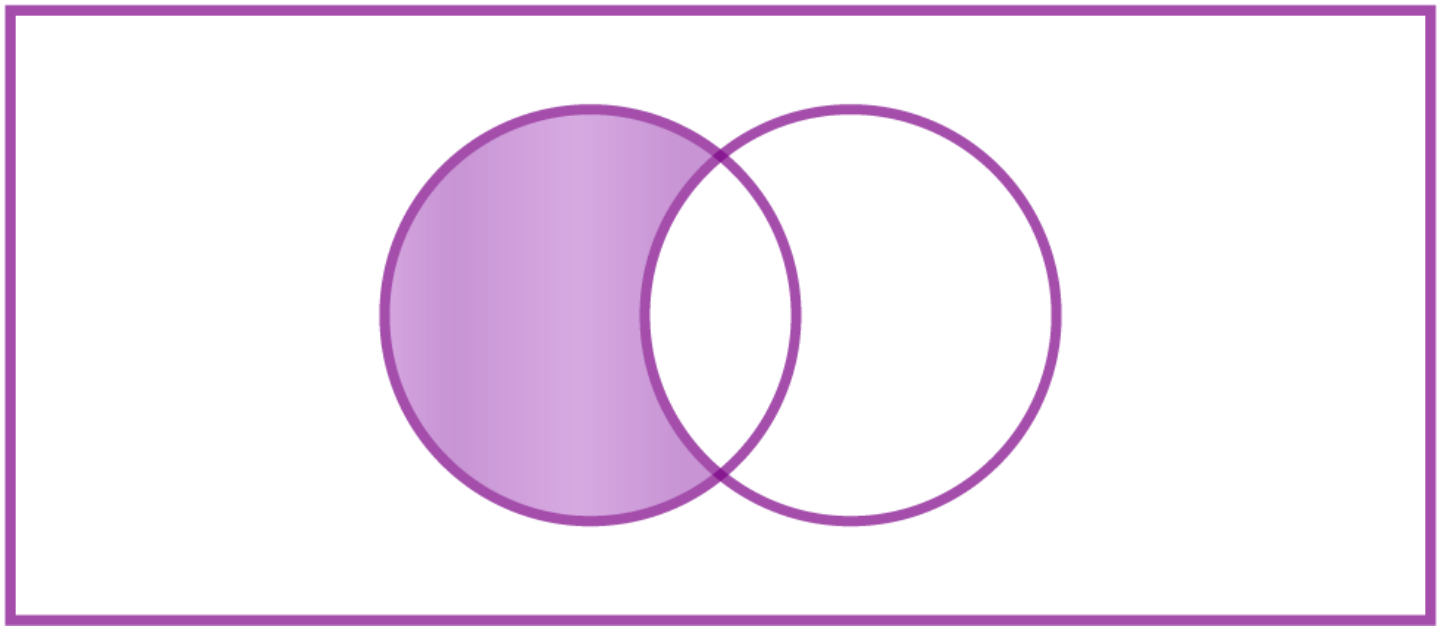
1. Объединение (union) — создает новое множество, содержащее все элементы из исходных множеств. Обозначается оператором |.



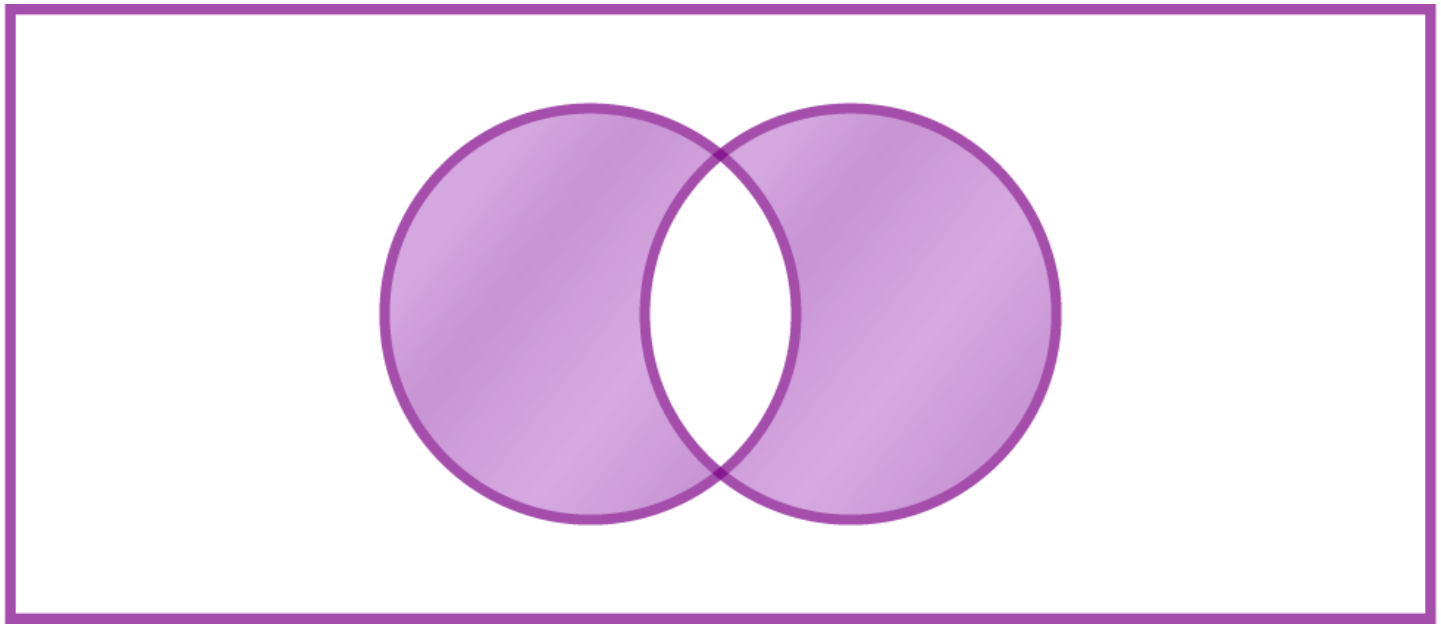
2. Пересечение (intersection) — создает новое множество, содержащее только общие элементы исходных множеств. Обозначается оператором &.



3. Разность (difference) — создает новое множество, содержащее элементы первого множества, не входящие во второе. Обозначается оператором -.



4. Симметрическая разность (symmetric difference) — создает новое множество, содержащее элементы, присутствующие в одном из множеств, но не в обоих. Обозначается оператором ^.



5. Проверка на подмножество (subset) — проверяет, являются ли все элементы одного множества элементами другого. Обозначается оператором <=.

6. Проверка на надмножество (superset) — проверяет, содержит ли одно множество все элементы другого. Обозначается оператором >=.
7. Добавление элемента (add) — добавляет элемент в множество.
8. Удаление элемента (remove) — удаляет элемент из множества.
9. Очистка множества (clear) — удаляет все элементы из множества.
10. Копирование множества (copy) — создает копию множества.

Примеры операций с множествами:

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

# Объединение
print(set1 | set2)  # {1, 2, 3, 4, 5}

# Пересечение
print(set1 & set2)  # {3}

# Разность
print(set1 - set2)  # {1, 2}

# Симметрическая разность
print(set1 ^ set2)  # {1, 2, 4, 5}

# Проверка на подмножество
print(set1 <= set2)  # False

# Проверка на надмножество
print(set1 >= set2)  # False

# Добавление элемента
set1.add(6)
print(set1)  # {1, 2, 3, 6}

# Удаление элемента
set1.remove(2)
print(set1)  # {1, 3, 6}

# Очистка множества
set1.clear()
print(set1)  # set()

# Копирование множества
set3 = set2.copy()
print(set3)  # {3, 4, 5}
```

Использование множеств в реальном коде:

1. Удаление дубликатов из списка:

```
my_list = [1, 2, 2, 3, 3, 3]
my_set = set(my_list)
my_list = list(my_set)
print(my_list)  # [1, 2, 3]
```

2. Проверка наличия элемента в списке:

```
my_list = [1, 2, 3, 4, 5]
my_set = set(my_list)

# Проверка наличия элемента в списке
if 3 in my_set:
    print("Элемент 3 присутствует в списке")
```

```
else:
    print("Элемент 3 отсутствует в списке")
```

3. Объединение списков без дубликатов:

```
list1 = [1, 2, 3]
list2 = [3, 4, 5]

set1 = set(list1)
set2 = set(list2)

result = list(set1 | set2)
print(result)  # [1, 2, 3, 4, 5]
```

4. Нахождение общих элементов в списках:

```
list1 = [1, 2, 3]
list2 = [3, 4, 5]

set1 = set(list1)
set2 = set(list2)

result = list(set1 & set2)
print(result)  # [3]
```

5. Нахождение элементов, отсутствующих во втором списке:

```
list1 = [1, 2, 3]
list2 = [3, 4, 5]

set1 = set(list1)
set2 = set(list2)

result = list(set1 - set2)
print(result)  # [1, 2]
```

6. Нахождение элементов, присутствующих только в одном из списков:

```
list1 = [1, 2, 3]
list2 = [3, 4, 5]

set1 = set(list1)
set2 = set(list2)

result = list(set1 ^ set2)
print(result)  # [1, 2, 4, 5]
```

7. Проверка на пересечение множеств:

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

if set1 & set2:
    print("Множества пересекаются")
else:
    print("Множества не пересекаются")
```

8. Проверка на вхождение подмножества:

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

```
if set2 ⩽ set1:
    print("set2 является подмножеством set1")
else:
    print("set2 не является подмножеством set1")
```

9. Проверка на вхождение надмножества:

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

if set1 ⩾ set2:
    print("set1 является надмножеством set2")
else:
    print("set1 не является надмножеством set2")
```

10. Нахождение всех подмножеств множества:

```
def find_subsets(s):
    return [set(comb) for comb in itertools.chain.from_iterable(itertools.combinations(s, r) for r in
range(len(s)+1))]

set1 = {1, 2, 3}

subsets = find_subsets(set1)
print(subsets)
# [{}, {1}, {2}, {3}, {1, 2}, {1, 3}, {2, 3}, {1, 2, 3}]
```

11. Нахождение всех надмножеств множества:

```
def find_supersets(s, universe):
    return [set(comb) for comb in itertools.chain.from_iterable(itertools.combinations(universe, r) for
r in range(len(universe)+1)) if s ⩽ set(comb)]

set1 = {1, 2}
universe = {1, 2, 3, 4}

supersets = find_supersets(set1, universe)
print(supersets)
# [{1, 2}, {1, 2, 3}, {1, 2, 4}, {1, 2, 3, 4}]
```

Заключение:

Множества в Python — это мощный инструмент для работы с наборами уникальных значений. Они позволяют выполнять операции, связанные с теорией множеств, такие как объединение, пересечение, разность и т.д. Кроме того, множества могут быть использованы для удаления дубликатов из списков, проверки наличия элемента в списке, объединения списков без дубликатов, нахождения общих элементов в списках и т.д.

Используя множества, вы можете делать свой код более эффективным и читабельным. Они позволяют решать многие задачи, связанные с обработкой данных, проще и быстрее. Поэтому изучение множеств в Python является важной частью обучения языку программирования Python.

Практика

Задача 1:

Напишите программу для вывода общего количества уникальных символов во всех считанных словах без учета регистра.

Формат входных данных
На вход программе в первой строке подается число n – общее количество слов. Далее идут n строк со словами.

Формат выходных данных
Программа должна вывести одно число – общее количество уникальных символов во всех словах без учета регистра.

Sample Input 1:
5
aAa
bB
ccc
dDdd
ppppP
Sample Output 1:
5

Sample Input 2:
4
авТорИтет
небо
машинА
Мёд
Sample Output 2:
13

Sample Input 3:
4
троС
рОст
сорт
тОрС
Sample Output 3:
4

Решение:

```
n = int(input())

# Создаем пустое множество для хранения уникальных символов
unique_chars = set()

# Считываем n слов и добавляем их уникальные символы в множество
for i in range(n):
    word = input().lower()
    for char in word:
        unique_chars.add(char)

# Выводим количество уникальных символов
print(len(unique_chars))
```

В этой программе мы сначала считываем количество слов `n`. Затем создаем пустое множество `unique_chars` для хранения уникальных символов. Далее считываем каждое слово, переводим его в нижний регистр, проходим по каждому символу слова и добавляем его в множество `unique_chars` с помощью метода `add()`. После того, как мы обработали все слова, выводим количество уникальных символов с помощью функции `len()`.

Эта программа решает задачу эффективно, так как множества в Python позволяют хранить только уникальные элементы и имеют быстрый поиск и добавление элементов.

Задача 2:

На вход программе подается строка текста, содержащая числа. Для каждого числа выведите слово `YES` (в отдельной строке), если это число ранее встречалось в последовательности или `NO`, если не встречалось.

Формат входных данных
На вход программе подается строка текста, содержащая числа, разделенные символом пробела.

Формат выходных данных
Программа должна вывести текст в соответствии с условием задачи.

Примечание. Ведущие нули в числах должны игнорироваться.

Sample Input 1:
1 1 2 2 5 5 5 6 7 8
Sample Output 1:
NO
YES
NO
YES
NO
YES
YES
YES
NO
NO
NO

Sample Input 2:
10 5 48 6 38 1
Sample Output 2:
NO
NO
NO
NO
NO
NO

Sample Input 3:
1 1 1 1 1 1 1 1 1
Sample Output 3:
NO
YES
YES
YES
YES
YES
YES
YES
YES

Sample Input 4:
1 2 3 002 03 4 5 05
Sample Output 4:
NO
NO
NO
YES
YES
NO
NO
YES

Решение:

```
numbers = list(map(int, input().split()))
seen_numbers = set()

for number in numbers:
    if number in seen_numbers:
        print("YES")
    else:
        seen_numbers.add(number)
        print("NO")
```

В этой программе мы сначала считываем строку текста, содержащую числа, разделенные символом пробела, и преобразуем ее в список чисел с помощью функции `map()` и метода `split()`. Затем создаем пустое множество `seen_numbers` для хранения уже встреченных чисел. Далее проходим по каждому числу в списке и проверяем, встречалось ли оно ранее с помощью оператора `in`. Если число уже встречалось, выводим слово `YES`, иначе добавляем его в множество `seen_numbers` с помощью метода `add()` и выводим слово `NO`.

В этой программе мы используем множество для хранения уже встреченных чисел, так как множества в Python позволяют хранить только уникальные элементы и имеют быстрый поиск. Это позволяет нам эффективно решать задачу.