Множества

Множество (set) представляют еще один вид набора, который хранит только уникальные элементы. Для определения множества используются фигурные скобки, в которых перечисляются элементы:

```
users = {"Tom", "Bob", "Alice", "Tom"}

print(users)    # {"Alice", "Bob", "Tom"}
```

Обратите внимание, что несмотря на то, что функция print вывела один раз элемент "Tom", хотя в определении множества этот элемент содержится два раза. Все потому что множество содержит только уникальные значения.

Также для определения множества может применяться функция set(), в которую передается список или кортеж элементов:

```
people = ["Mike", "Bill", "Ted"]

users = set(people)

print(users)    # {"Mike", "Bill", "Ted"}
```

Функцию set удобно применять для создания пустого множества:

```
users = set()
```

Для получения длины множества применяется встроенная функция len():

```
users = {"Tom", "Bob", "Alice"}

print(len(users))       # 3
```

Добавление элементов
Для добавления одиночного элемента вызывается метод add():

```
users = set()

users.add("Sam")

print(users)
```

Удаление элементов
Для удаления одного элемента вызывается метод remove(), в который передается удаляемый элемент. Но следует учитывать, что если такого элемента не окажется в множестве, то будет сгенерирована ошибка. Поэтому перед удалением следует проверять на наличие элемента с помощью оператора in:

```
users = {"Tom", "Bob", "Alice"}

user = "Tom"

if user in users:

   users.remove(user)

print(users)    # {"Bob", "Alice"}
```

Также для удаления можно использовать метод discard(), который не будет генерировать исключения при отсутствии элемента:

```
users = {"Tom", "Bob", "Alice"}

users.discard("Tim")    # элемент "Tim" отсутствует, и метод ничего не делает
```

```
print(users)    #  {"Tom", "Bob", "Alice"}

users.discard("Tom")    # элемент "Tom" есть, и метод удаляет элемент

print(users)    #  {"Bob", "Alice"}
```

Для удаления всех элементов вызывается метод clear():

```
users.clear()
```

Перебор множества
Для перебора элементов можно использовать цикл for:

```
users = {"Tom", "Bob", "Alice"}

for user in users:

   print(user)

#При переборе каждый элемент помещается в переменную user.
```

Операции с множествами
С помощью метода copy() можно скопировать содержимое одного множества в другую переменную:

```
users = {"Tom", "Bob", "Alice"}

students = users.copy()

print(students)     # {"Tom", "Bob", "Alice"}
```

Метод union() объединяет два множества и возвращает новое множество:

```
users = {"Tom", "Bob", "Alice"}

users2 = {"Sam", "Kate", "Bob"}

users3 = users.union(users2)

print(users3)   # {"Bob", "Alice", "Sam", "Kate", "Tom"}
```

Пересечение множеств позволяет получить только те элементы, которые есть одновременно в обоих множествах. Метод intersection() производит операцию пересечения множеств и возвращает новое множество:

```
users = {"Tom", "Bob", "Alice"}

users2 = {"Sam", "Kate", "Bob"}

users3 = users.intersection(users2)

print(users3)   # {"Bob"}
```

Вместо метода intersection мы могли бы использовать операцию логического умножения:

```
users = {"Tom", "Bob", "Alice"}

users2 = {"Sam", "Kate", "Bob"}

print(users & users2)   # {"Bob"}
```

В этом случае мы получили бы тот же результат.

Еще одна операция - разность множеств возвращает те элементы, которые есть в первом множестве, но отсутствуют во втором. Для получения разности множеств можно использовать метод difference или операцию вычитания:

```
users = {"Tom", "Bob", "Alice"}

users2 = {"Sam", "Kate", "Bob"}

users3 = users.difference(users2)

print(users3)          # {"Tom", "Alice"}

print(users - users2)  # {"Tom", "Alice"}
```

Отдельная разновидность разности множеств - симметрическая разность производится с помощью метода symmetric_difference(). Она возвращает все элементы обоих множеств за исключением общих:

```
users = {"Tom", "Bob", "Alice"}

users2 = {"Sam", "Kate", "Bob"}

users3 = users.symmetric_difference(users2)

print(users3)  # {"Tom", "Alice", "Sam", "Kate"}
```

Отношения между множествами
Метод issubset позволяет выяснить, является ли текущее множество подмножеством (то есть частью) другого множества:

```
users = {"Tom", "Bob", "Alice"}

superusers = {"Sam", "Tom", "Bob", "Alice", "Greg"}

print(users.issubset(superusers))  # True

print(superusers.issubset(users))  # False
```

Метод issuperset, наоборот, возвращает True, если текущее множество является надмножеством (то есть содержит) для другого множества:

```
users = {"Tom", "Bob", "Alice"}

superusers = {"Sam", "Tom", "Bob", "Alice", "Greg"}

print(users.issuperset(superusers))   # False

print(superusers.issuperset(users))   # True
```

frozen set
Тип frozen set является видом множеств, которое не может быть изменено. Для его создания используется функция frozenset:

```
users = frozenset({"Tom", "Bob", "Alice"})
```

В функцию frozenset передается набор элементов - список, кортеж, другое множество.

В такое множество мы не можем добавить новые элементы, как и удалить из него уже имеющиеся. Собственно поэтому frozen set поддерживает ограниченный набор операций:

len(s): возвращает длину множества

x in s: возвращает True, если элемент x присутствует в множестве s

x not in s: возвращает True, если элемент x отсутствует в множестве s

s.issubset(t): возвращает True, если t содержит множество s

s.issuperset(t): возвращает True, если t содержится в множестве s

s.union(t)

: возвращает объединение множеств s и t
s.intersection(t): возвращает пересечение множеств s и t

s.difference(t): возвращает разность множеств s и t

s.copy(): возвращает копию множества s

_____
_____
_____