Оглавление

Задачи

Множества (set) и их методы

На этом занятии мы с вами познакомимся с последней базовой коллекцией – множествами.

Формально, множество (set) – это неупорядоченная коллекция уникальных элементов. Уникальных, то есть, в ней отсутствуют дублирующие значения.

Для задания множества используются фигурные скобки, в которых через запятую перечисляются значения элементов: `a = {1, 2, 3, "hello"}`

Это немного похоже на определение словаря, но в отличие от него, здесь не прописываются ключи – только значения. В результате получаем совершенно другой тип объекта – множество:

Так как множество состоит только из уникальных значений, то попытка в него поместить несколько одинаковых значений:

```
a = {1, 2, 3, "hello", 2, 3, "hello"}
```

приведет к тому, что останутся только не повторяющиеся. Это ключевая особенность работы данной коллекции – она автоматически отбрасывает все дубли.

Во множество мы можем записывать только неизменяемые данные, такие как:

- числа;
- булевы значения;
- строки;
- кортежи

```
a = {1, 4.5, "hi", "hi", (4, True), ("a", False)}
```

А вот изменяемые типы данных: добавлять нельзя, будет ошибка:

- списки;
- словари;
- множества

```
b = {[1, 2], 3, 4}
b = {{1: 1, 2: 2}, 3, 4}
b = {{1, 2}, 3, 4}
```

Для создания множества используется встроенная функция set().
Если ее записать без аргументов: `set()`

то получим пустое множество. Но, обратите внимание, если попытаться создать пустое множество вот так:

```
b = {}  # словарь
```

то получим не пустое множество, а пустой словарь! Пустое множество создается именно с помощью функции set(). Вот этот момент нужно запомнить.

Если указать какой-либо итерируемый объект, например, список:

```
set([1, 2, 1, 0, -1, 2])
```

то получим множество, состоящее из уникальных значений этого списка. Или, если передать строку: `set("abrakadabra")`

то увидим множество из уникальных символов этой строки. И так для любого итерируемого объекта, даже можно прописать функцию range():

```
b = set(range(7))
```

Обратите внимание, множества представляют собой неупорядоченную коллекцию, то есть, значения элементов могут располагаться в них в любом порядке. Это также означает, что с множествами нельзя выполнять операции доступа по индексу или срезам: `b[0]`

приведет к ошибке.

Для чего вообще может потребоваться такая коллекция в программировании? Я, думаю, большинство из вас уже догадались –с их помощью, например, можно легко и быстро убирать дубли из данных. Представим, что некоторая логистическая фирма составила список городов, куда доставляли товары:

```
cities = ["Калуга", "Краснодар", "Тюмень", "Ульяновск",
"Москва", "Тюмень", "Калуга", "Ульяновск"]
```

И отдел статистики хотел бы получить список уникальных городов, с которыми было сотрудничество. Очевидно, это можно сделать с помощью множества:

```
set(cities)
```

Обратите внимание, порядок городов может быть любым, так как множество – неупорядоченная коллекция данных. Мало того, мы можем этот полученный перечень снова преобразовать в список, используя функцию: `list(set(cities))`

Все, мы отобрали уникальные города и представили их в виде списка. Видите, как это легко и просто можно сделать с помощью множеств.

А что если нам не нужен список уникальных городов, а достаточно лишь перебрать все элементы полученного множества: `q = set(cities)`

Учитывая, что множество – это итерируемый объект, то пройтись по всем его элементам можно с помощью цикла for:

```
for c in q:
    print(c)
```

Здесь переменная с последовательно ссылается на значения элементов множества q и соответствующий город выводится в консоль.

Наконец, функция:

```
a = {"abc", (1, 2), 5, 4, True}
len(a)
```

возвратит нам число элементов в множестве. А для проверки наличия значения в множестве можно воспользоваться, уже знакомым нам оператором in: `"abc" in a`

Методы добавления/удаления элементов множества

В заключение этого занятия рассмотрим методы для добавления и удаления элементов в множествах. Первый метод add() позволяет добавлять новое значение в множество: `b.add(7)`

Значение будет добавлено только в том случае, если оно отсутствует в множестве b. Если попробовать, например, еще раз добавить семерку: `b.add(7)`

то множество не изменится, но и ошибок никаких не будет.

Если в множество требуется добавить сразу несколько значений, то для этого хорошо подходит метод update(): `b.update(["a", "b", (1, 2)])`

В качестве аргумента указывается любой итерируемый объект, в данном случае, список, значения которого добавляются в множество с проверкой уникальности. Или, можно передать строку: `b.update("abrakadabra")`

Получим добавку в виде уникальных символов этой строки. И так далее, в качестве аргумента метода update() можно указывать любой перебираемый объект. Далее, для удаления элемента по значению используется метод discard(): `b.discard(2)`

Другой метод для удаления элемента по значению – remove(): `b.remove(4)`

но при повторном таком вызове: `b.remove(4)`

возникнет ошибка, т.к. значение 4 в множестве уже нет. То есть, данный метод возвращает ошибку при попытке удаления несуществующего значения. Это единственное отличие в работе этих двух методов.

Также удалять элементы можно и с помощью метода pop: `b.pop()`

При этом он возвращает удаляемое значение, а сам удаляемый элемент оказывается, в общем-то, случайным, т.к. множество – это неупорядоченная коллекция. Если вызвать этот метод для пустого множества, то возникнет ошибка:

Наконец, если нужно просто удалить все элементы из множества, то используется метод: `b.clear()`