

# Лабораторная работа №6

## Множества. Словари.

### 1. Множества

Множество в языке Питон — это структура данных, эквивалентная множествам в математике. Множество может состоять из различных элементов, порядок элементов в множестве неопределен. В множество можно добавлять и удалять элементы, можно перебирать элементы множества, можно выполнять операции над множествами (объединение, пересечение, разность). Можно проверять принадлежность элемента множеству.

В отличие от массивов, где элементы хранятся в виде последовательного списка, в множествах порядок хранения элементов неопределен (более того, элементы множества хранятся не подряд, как в списке, а при помощи хитрых алгоритмов). Это позволяет выполнять операции типа “проверить принадлежность элемента множеству” быстрее, чем просто перебирая все элементы множества.

Элементами множества может быть любой неизменяемый тип данных: числа, строки, кортежи. Изменяемые типы данных не могут быть элементами множества, в частности, нельзя сделать элементом множества список (но можно сделать кортеж) или другое множество. Требование неизменяемости элементов множества накладывается особенностями представления множества в памяти компьютера.

#### Задание множеств

Множество задается перечислением всех его элементов в фигурных скобках. Исключением является пустое множество, которое можно создать при помощи функции `set()`. Если функции `set` передать в качестве параметра список, строку или кортеж, то она вернёт множество, составленное из элементов списка, строки, кортежа. Например:

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

выведет `{'e', 'q', 'r', 't', 'w', 'y'}`.

Каждый элемент может входить в множество только один раз, порядок задания элементов неважен. Например, программа:

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

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'}`.

#### Работа с элементами множеств

Узнать число элементов в множестве можно при помощи функции `len`.

Перебрать все элементы множества (в неопределенном порядке!) можно при помощи цикла `for`:

```
primes = {2, 3, 5, 7, 11}
for num in primes:
    print(num)
```

Проверить, принадлежит ли элемент множеству можно при помощи операции `in`, возвращающей значение типа `bool`. Аналогично есть противоположная операция `not in`. Для добавления элемента в множество есть метод `add`:

```
A = {1, 2, 3}
print(1 in A, 4 not in A)
A.add(4)
```

Для удаления элемента  $x$  из множества есть два метода: `discard` и `remove`. Их поведение различается только в случае, когда удаляемый элемент отсутствует в множестве. В этом случае метод `discard` не делает ничего, а метод `remove` генерирует исключение `KeyError`.

Наконец, метод `pop` удаляет из множества один случайный элемент и возвращает его значение. Если же множество пусто, то генерируется исключение `KeyError`.

Из множества можно сделать список при помощи функции `list`.

### Операции с множествами

С множествами в питоне можно выполнять обычные для математики операции над множествами.

$A \cup B$ <code>A.union(B)</code>	Возвращает множество, являющееся объединением множеств $A$ и $B$ .
$A \cup= B$ <code>A.update(B)</code>	Добавляет в множество $A$ все элементы из множества $B$ .
$A \cap B$ <code>A.intersection(B)</code>	Возвращает множество, являющееся пересечением множеств $A$ и $B$ .
$A \cap= B$ <code>A.intersection_update(B)</code>	Оставляет в множестве $A$ только те элементы, которые есть в множестве $B$ .
$A - B$ <code>A.difference(B)</code>	Возвращает разность множеств $A$ и $B$ (элементы, входящие в $A$ , но не входящие в $B$ ).
$A -= B$ <code>A.difference_update(B)</code>	Удаляет из множества $A$ все элементы, входящие в $B$ .
$A \Delta B$ <code>A.symmetric_difference(B)</code>	Возвращает симметрическую разность множеств $A$ и $B$ (элементы, входящие в $A$ или в $B$ , но не в оба из них одновременно).
$A \Delta= B$ <code>A.symmetric_difference_update(B)</code>	Записывает в $A$ симметрическую разность множеств $A$ и $B$ .
$A \leq B$ <code>A.issubset(B)</code>	Возвращает <code>true</code> , если $A$ является подмножеством $B$ .
$A \geq B$ <code>A.issuperset(B)</code>	Возвращает <code>true</code> , если $B$ является подмножеством $A$ .
$A < B$	Эквивалентно $A \leq B$ and $A \neq B$
$A > B$	Эквивалентно $A \geq B$ and $A \neq B$

## 2. Словари

Обычные списки (массивы) представляют собой набор пронумерованных элементов, то есть для обращения к какому-либо элементу списка необходимо указать его номер. Номер элемента в списке однозначно идентифицирует сам элемент. Но идентифицировать данные по числовым номерам не всегда оказывается удобно. Например, маршруты поездов в России идентифицируются численно-буквенным кодом (число и одна буква), также численно-буквенным кодом идентифицируются авиарейсы, то есть для хранения информации о рейсах поездов или самолетов в качестве идентификатора удобно было бы использовать не число, а текстовую строку.

Структура данных, позволяющая идентифицировать ее элементы не по числовому индексу, а по произвольному, называется **словарем** или ассоциативным массивом. Соответствующая структура данных в языке Питон называется `dict`.

Рассмотрим простой пример использования словаря. Заведем словарь `Capitals`, где индексом является название страны, а значением — название столицы этой страны. Это позволит легко определять по строке с названием страны ее столицу.

```
Capitals = dict()

# Заполним его несколькими значениями
Capitals['Russia'] = 'Moscow'
Capitals['Ukraine'] = 'Kiev'
Capitals['USA'] = 'Washington'

Countries = ['Russia', 'France', 'USA', 'Russia']

for country in Countries:
    # Для каждой страны из списка проверим, есть ли она в словаре Capitals
    if country in Capitals:
        print('Столица страны ' + country + ': ' + Capitals[country])
    else:
        print('В базе нет страны с названием ' + country)
```

Итак, каждый элемент словаря состоит из двух объектов: ключа и значения. В нашем примере ключом является название страны, значением является название столицы. Ключ идентифицирует элемент словаря, значение является данными, которые соответствуют данному ключу. Значения ключей — уникальны, двух одинаковых ключей в словаре быть не может.

В жизни широко распространены словари, например, привычные бумажные словари (толковые, орфографические, лингвистические). В них ключом является слово-заголовок статьи, а значением — сама статья. Для того, чтобы получить доступ к статье, необходимо указать слово-ключ.

Другой пример словаря, как структуры данных — телефонный справочник. В нем ключом является имя, а значением — номер телефона. И словарь, и телефонный справочник хранятся так, что легко найти элемент словаря по известному ключу (например, если записи хранятся в алфавитном порядке ключей, то легко можно найти известный ключ, например, бинарным поиском), но если ключ неизвестен, а известно лишь значение, то поиск элемента с данным значением может потребовать последовательного просмотра всех элементов словаря.

Особенностью ассоциативного массива является его динамичность: в него можно добавлять новые элементы с произвольными ключами и удалять уже существующие элементы. При этом размер используемой памяти пропорционален размеру ассоциативного массива. Доступ к элементам ассоциативного массива выполняется хоть и медленнее, чем к обычным массивам, но в целом довольно быстро.

В языке Питон ключом может быть произвольный неизменяемый тип данных: целые и действительные числа, строки, кортежи. Ключом в словаре не может быть множество, но может быть элемент типа `frozenset`: специальный тип данных, являющийся аналогом типа `set`, который нельзя изменять после создания. Значением элемента словаря может быть любой тип данных, в том числе и изменяемый.

Словари нужно использовать в следующих случаях:

Подсчет числа каких-то объектов. В этом случае нужно завести словарь, в котором ключами являются объекты, а значениями — их количество.

Хранение каких-либо данных, связанных с объектом. Ключи — объекты, значения — связанные с ними данные. Например, если нужно по названию месяца определить его порядковый номер, то это можно сделать при помощи словаря `Num['January'] = 1; Num['February'] = 2; ....`

Установка соответствия между объектами (например, “родитель—потомок”). Ключ — объект, значение — соответствующий ему объект.

Если нужен обычный массив, но максимальное значение индекса элемента очень велико, и при этом будут использоваться не все возможные индексы (так называемый “разреженный массив”), то можно использовать ассоциативный массив для экономии памяти.

#### Создание словаря

Пустой словарь можно создать при помощи функции `dict()` или пустой пары фигурных скобок `{}` (вот почему фигурные скобки нельзя использовать для создания пустого множества). Для создания словаря с некоторым набором начальных значений можно использовать следующие конструкции:

```
Capitals = {'Russia': 'Moscow', 'Ukraine': 'Kiev', 'USA': 'Washington'}
Capitals = dict(Russia = 'Moscow', Ukraine = 'Kiev', USA = 'Washington')
Capitals = dict([("Russia", "Moscow"), ("Ukraine", "Kiev"), ("USA",
"Washington")])
Capitals = dict(zip(["Russia", "Ukraine", "USA"], ["Moscow", "Kiev",
"Washington"]))
print(Capitals)
```

Первые два способа можно использовать только для создания небольших словарей, перечисляя все их элементы. Кроме того, во втором способе ключи передаются как именованные параметры функции `dict`, поэтому в этом случае ключи могут быть только строками, причем являющимися корректными идентификаторами. В третьем и четвертом случае можно создавать большие словари, если в качестве аргументов передавать уже готовые списки, которые могут быть получены не обязательно перечислением всех элементов, а любым другим способом построены по ходу исполнения программы. В третьем способе функции `dict` нужно передать список, каждый элемент которого является кортежем из двух элементов: ключа и значения. В четвертом способе используется функция `zip`, которой передаются два списка одинаковой длины: список ключей и список значений.

#### Работа с элементами словаря

Основная операция: получение значения элемента по ключу, записывается так же, как и для списков: `A[key]`. Если элемента с заданным ключом нет в словаре, то возникает исключение `KeyError`.

Другой способ определения значения по ключу — метод `get`: `A.get(key)`. Если элемента с ключом `get` нет в словаре, то возвращается значение `None`. В форме записи с двумя аргументами `A.get(key, val)` метод возвращает значение `val`, если элемент с ключом `key` отсутствует в словаре.

Проверить принадлежность элемента словарю можно операциями `in` и `not in`, как и для множеств.

Для добавления нового элемента в словарь нужно просто присвоить ему какое-то значение: `A[key] = value`.

Для удаления элемента из словаря можно использовать операцию `del A[key]` (операция возбуждает исключение `KeyError`, если такого ключа в словаре нет. Вот два безопасных способа удаления элемента из словаря.

```
A = {'ab' : 'ba', 'aa' : 'aa', 'bb' : 'bb', 'ba' : 'ab'}

key = 'ac'
if key in A:
    del A[key]

try:
    del A[key]
except KeyError:
    print('There is no element with key "' + key + '" in dict')
print(A)
```

В первом случае мы предварительно проверяем наличие элемента, а во втором - перехватываем и обрабатываем исключение.

Еще один способ удалить элемент из словаря: использование метода `pop`: `A.pop(key)`. Этот метод возвращает значение удаляемого элемента, если элемент с данным ключом отсутствует в словаре, то возбуждается исключение. Если методу `pop` передать второй параметр, то если элемент в словаре отсутствует, то метод `pop` возвратит значение этого параметра. Это позволяет проще всего организовать безопасное удаление элемента из словаря: `A.pop(key, None)`.

### Перебор элементов словаря

Можно легко организовать перебор ключей всех элементов в словаре:

```
A = dict(zip('abcdef', list(range(6))))
for key in A:
    print(key, A[key])
```

Следующие методы возвращают представления элементов словаря. Представления во многом похожи на множества, но они изменяются, если менять значения элементов словаря. Метод `keys` возвращает представление ключей всех элементов, метод `values` возвращает представление всех значений, а метод `items` возвращает представление всех пар (кортежей) из ключей и значений.

Соответственно, быстро проверить, есть ли значение `val` среди всех значений элементов словаря `A` можно так: `val in A.values()`, а организовать цикл так, чтобы в переменной `key` был ключ элемента, а в переменной `val`, было его значение можно так:

```
A = dict(zip('abcdef', list(range(6))))
for key, val in A.items():
    print(key, val)
```

## 3. Практические задания

**К любой задаче по требованию преподавателя уметь строить блок-схемы.**

1. Дан список чисел, введенных с клавиатуры. Определить, сколько раз встречаются различные цифры.
2. Даны два списка чисел, введенных с клавиатуры. Определить, сколько и каких чисел одновременно встречается в двух списках.
3. Дан текст, состоящий из строк. Определить количество слов в тексте. Словом считается последовательность символов, слова разделены пробелом или символом конца строки. Используйте множества.

4. Каждый из некоторого множества школьников некоторой школы знает некоторое количество языков. Нужно определить сколько языков знают все школьники, и сколько языков знает хотя бы один из школьников.  
В первой строке задано количество школьников. Для каждого из школьников сперва записано количество языков, которое он знает, а затем - названия языков, по одному в строке.  
В первой строке выведите количество языков, которые знают все школьники. Начиная со второй строки - список таких языков. Затем - количество языков, которые знает хотя бы один школьник, на следующих строках - список таких языков. Языки нужно выводить в лексикографическом порядке, по одному на строке. Используйте множества.
5. Даны два множества чисел. Выведите третье, состоящее из элементов первого и элементов второго множества, которых нет в первом.
6. Дан текст, состоящий из количества строк текста и самого текста. Вывести на экран построчно каждое слово и сколько раз оно встречается в тексте. Для реализации используйте словари.
7. Дан текст, состоящий из количества строк текста и самого текста. Вывести на экран самое длинное слово. Для реализации используйте словари.
8. Дан текст, состоящий из количества строк текста и самого текста. Вывести на экран слово, которое встречается в тексте чаще всего. Для реализации используйте словари.
9. Имеются данные о продажах спортивного интернет-магазина. Каждая строка представлена записью ФИО покупателя[пробел]наименование товара[пробел]Количество товара. Выведите список всех покупателей, в котором для каждого покупателя подсчитайте количество приобретенных им единиц каждого вида товаров. Например:  

На входе:	На выходе:
Иванов лыжи 1	Иванов:
Петров коньки 3	лыжи 1
Иванов сумка 1	сумка 1
Иванов палки 2	палки 2
Петров куртка 1	Петров:
	коньки 3
	куртка 1
10. Имеется список областей России и городов, относящихся к области. Определить, к какой области относится город. Например:  
На входе:  
1  
Архангельская область: Архангельск, Новодвинск, Северодвинск, Шенкурск, Котласс.  
Ленинградская область: Санкт-Петербург, Пушкин, Павловск.  
  
2  
Архангельск  
Пушкин  
Котласс  
  
На выходе  
Архангельская область  
Ленинградская область  
Архангельская область