

# Глава 4 Коллекции

Используя скалярные типы можно столкнуться с проблемой – что делать, если необходимо хранить и обрабатывать набор таких значений? Для этого в Python предназначены специальные типы – **коллекции**.

**Коллекция** – это переменная-контейнер, в которой может содержаться какое-то количество объектов, эти объекты могут быть одного типа или разного.

В коллекциях **поддерживается**:

- проверка на вхождения элементов `in` и `not in` (`True/False`);
- определение размера `len()`;
- возможность выполнения итераций (перемещения по элементам последовательности) – из-за этого коллекции также называются итерируемыми объектами.

**Итерируемый объект** – это объект, позволяющий поочерёдно перебирать составляющие его элементы.

Среди коллекций выделяют 3 группы:

- **последовательности**: строка (`str`), юникод-строки (`unicode`), список (`list`), кортеж (`tuple`), числовой диапазон (`range`);
- **отображения**: словарь;
- **множества**.

## § 4.1 Последовательности. Общие операции для последовательностей

**Последовательность** – это упорядоченная коллекция, поддерживающая индексированный доступ к элементам.

Некоторые последовательности в Python в отличие от традиционных массивов (например, в Паскале или Си) могут хранить элементы различного типа (в том числе и коллекции различных видов).

В языке Python имеется пять встроенных типов последовательностей, среди которых: `str` (строка), `list` (список), `tuple` (кортеж) и `range` (числовой диапазон). Некоторые операции справедливы для всех последовательностей (за исключением случаев, где они не возможны исходя из определения типа).

Таблица. Общие методы для последовательностей

Название метода	Синтаксис метода	Описание метода
Длина	<code>len(s)</code>	Возвращает длину (количество элементов в последовательности) <code>s</code>
Конкатенация «склеивание»	<code>s + t</code>	Возвращает новый объект – «склейку» <code>s</code> и <code>t</code>

Название метода	Синтаксис метода	Описание метода
Дублирование	<code>s * n ( n * s )</code>	Возвращает последовательность <code>s</code> , повторяющуюся <code>n</code> раз
Индексация и срезы	<code>s[start]</code> <code>s[start:end]</code> <code>s[start:end:step]</code>	Получить доступ к отдельному или группе элементов последовательности. Индексацию (получение отдельного элемента) можно считать частным случаем получения среза (слайсинга): <code>s[start]</code> – элемент с индексом <code>start</code> (индексация с 0); <code>s[start:end]</code> – срез <code>[start; end)</code> ; <code>s[start:end:step]</code> – срез <code>[start; end)</code> с шагом <code>step</code> . В ряде случаев целочисленные параметры <code>start</code> , <code>end</code> и <code>step</code> могут быть опущены. Элемент с индексом <code>end</code> не включается в результат при взятии срезов.
Минимальное значение	<code>min(s)</code>	Возвращает минимальный элемент последовательности <code>s</code>
Максимальное значение	<code>max(s)</code>	Возвращает максимальный элемент последовательности <code>s</code>
Проверка на вхождение	<code>x in s</code>	Возвращает <code>True</code> , если <code>x</code> входит в последовательность <code>s</code> и <code>False</code> в противном случае. Проверка на вхождение элемента в список происходит за линейное время
Индекс (положение) элемента	<code>s.index(x[, start[, end]])</code>	Возвращает первое вхождение элемента <code>x</code> в последовательность <code>s</code> (между индексами <code>start</code> и <code>end</code> , если они заданы). Результат – целое число
Количество повторений	<code>s.count(x)</code>	Возвращает количество вхождений элементов <code>x</code> в последовательность <code>s</code>
Сортировка	<code>sorted(iterable, key=None, reverse=False)</code>	Возвращает отсортированный объект в виде списка. <b>Исходный объект при этом не изменяется</b> . Параметры: <code>key</code> – функция сортировки (по умолчанию не учитывается, сортировка осуществляется поэлементно); <code>reverse</code> – если равен <code>True</code> , сортировка осуществляется в обратном порядке

В таблице используются следующие обозначения:

- `s` и `t` – последовательности одного типа;
- `n`, `k`, `start`, `end`, `step` – целые числа;
- `x` – произвольный объект, отвечающий критериям соответствующего вызова функции

**Пример** использования методов для работы с последовательностями.

В [1]:

```
# Использование методов для работы с последовательностями
ls = [12, 96, -97, 23, 2, 26, 45, 85, -6]
print('Номер максимального:', ls.index(max(ls)) + 1)

st = 'фу-фу-фу, фи-фи-фи'
print('Количество букв "ф" - ', st.count('ф'))
```

Номер максимального: 2

Количество букв "ф" - 6

В [2]:

```
ls = [34, 67, 89, 3, 96, 5, 9, -63, 96, 12, -6]
print('Номер 2-го максимума:', ls.index(max(ls)), ls.index(max(ls)) + 1) + 1
```

Номер второго максимума: 9

## § 4.2 Списки

## 4.2.1 Создание списков. Методы обработки списков

**Список ( list )** – это упорядоченная изменяемая последовательность элементов, которая 1) может содержать **элементы разного типа**, и 2) поддерживает операторы сравнения, при этом сравнивание производится поэлементно (и рекурсивно, при наличии вложенных элементов). Нумерация элементов в списке начинается **с нуля**.

У разработчиков типа данных `list` Python было много вариантов каким сделать его во время реализации. Каждый выбор повлиял на то, как быстро список мог выполнять операции. Одно из решений было сделать список оптимальным для частых операций.

Список в Python имеет обманчивое название. Это не связный список, а динамический массив, элементами которого могут выступать объекты произвольных типов.

Важно учитывать, что **списки являются изменяемыми** и передаются по ссылке, а не по значению, как показано в примере ниже.

В [3]:

```
x = y = []
x.append(1)
x.append(2)
print(x, y)
```

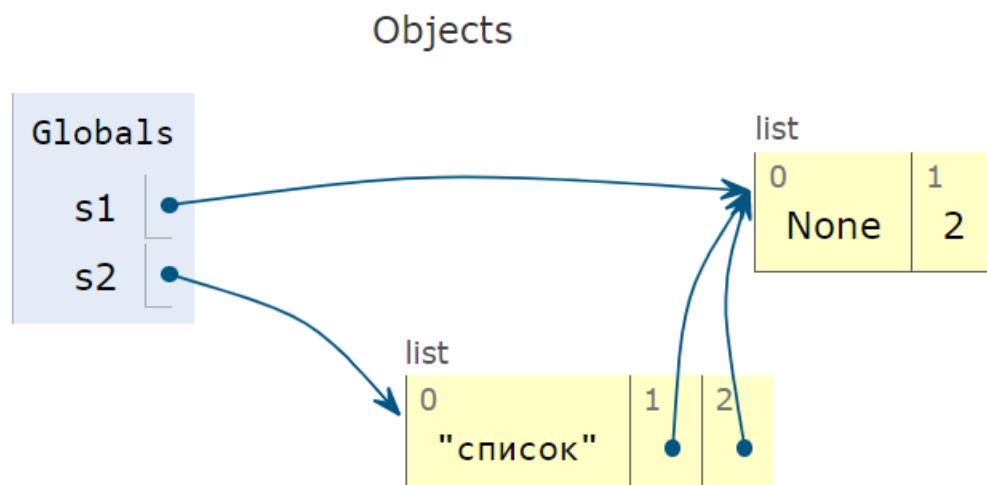
[1, 2] [1, 2]

В [4]:

```
s1 = [1, 2]
s2 = ['список', s1, s1]
print(s2)
s1[0] = None
print(s2)
```

['список', [1, 2], [1, 2]]
['список', [None, 2], [None, 2]]

Рассмотренная в примере ситуация изображена на картинке ниже.



Попробуйте поэкспериментировать со списками с помощью сайта [pythontutor.com](http://pythontutor.com) (<http://pythontutor.com>).

По возможности, следует использовать неизменяемые типы данных, поскольку, как видно в примере выше, модификация списка может привести к неожиданным результатам и ошибкам, которые, зачастую, трудно обнаружить. В частности, не следует модифицировать список в цикле прохода по элементам этого списка.

Для списков и других изменяемых последовательностей используются те же общие операции, что и для строк, а также иных неизменяемых последовательностей. Кроме того, существует ряд дополнительных методов, специально предназначенных для изменяемых последовательностей.

**Таблица.** Дополнительные методы для списков ( `lst` – список, на котором вызывается метод)

Операция	Описание	Трудоемкость
<code>lst[i]</code>	Взятие элемента по индексу	$O(1)$
<code>lst[i] = x</code>	Заменить элемент по индексу	$O(1)$
<code>lst1[i:j] = lst2</code> , <code>lst1[i:j:k] = lst2</code>	Переписать срез	
<code>del lst[i]</code> , <code>del lst[i:j]</code> , <code>del lst[i:j:k]</code>	Удалить элемент или срез	
<code>lst.append(x)</code>	Добавить в конец <code>lst</code> элемент <code>x</code>	$O(1)$
<code>lst.clear()</code>	Удалить все элементы из <code>lst</code>	
<code>lst[:]</code> , <code>lst.copy()</code>	Получить копию <code>lst</code> (копируются ссылки на объекты)	
<code>lst1 += lst2</code> , <code>lst1.extend(lst2)</code>	Добавляет в конец списка <code>lst1</code> все элементы коллекции <code>lst2</code>	
<code>lst.insert(i, x)</code>	Вставить <code>x</code> в <code>lst</code> по индексу <code>i</code>	
<code>lst.pop()</code>	Возвращает последний элемент, удаляя его из <code>lst</code>	$O(1)$
<code>lst.pop(i)</code>	Возвращает последний <code>i</code> -й элемент, удаляя его из <code>lst</code>	$O(n)$
<code>lst.remove(x)</code>	Удалить первое вхождение <code>x</code> из <code>lst</code>	
<code>lst.reverse()</code>	Обратить последовательность <code>lst</code>	$O(N)$
<code>lst.sort(*, key=None, reverse=None)</code>	Выполняет сортировку списка <code>lst</code> . Отличается от функции <code>sorted()</code> тем, что сортирует исходный объект, а не возвращает новый. Параметры: <code>key</code> – функция сортировки (по умолчанию не учитывается, сортировка осуществляется поэлементно); <code>reverse</code> – если равен <code>True</code> , сортировка осуществляется в обратном порядке	
<code>sum(lst)</code>	Возвращает сумму списка <code>lst</code>	

### Трудоемкость стандартных операций по работе со списком

Важно понять трудоемкость стандартных процедур в языке Python. Рассмотрим трудоемкость встроенных методов и операций класса `list`.

Следующие операции над списком имеют вычислительную сложность  $O(1)$ :

- `lst.append(x)`,
- `lst.pop()`,
- `lst[i]`,
- `lst[i] = x`,
- `len(lst)`.

**Индексирование и присваивание.** Две частые операции – индексирование и присваивание на позицию индекса. В списках Python значения присваиваются и извлекаются из определенных известных мест памяти. Независимо от того, насколько велик список, индексный поиск и присвоение занимают постоянное количество времени и, таким образом их трудоемкость  $O(1)$ .

**Pop, Delete.** Извлечение элемента ( `pop` ) из списка Python по умолчанию выполняется с конца, но, передавая индекс, вы можете получить элемент из определенной позиции. Когда `pop` вызывается с конца, операция имеет сложность  $O(1)$ , а вызов `pop` из любого места –  $O(n)$ . Откуда такая разница?

Когда элемент берется из середины списка Python, все остальные элементы в списке сдвигаются на одну позицию ближе к началу. Это суровая плата за возможность брать индекс за  $O(1)$ , что является более частой операцией. По тем же причинам вставка в индекс –  $O(N)$ ; каждый последующий элемент должен быть сдвинут на одну позицию ближе к концу, чтобы разместить новый элемент. Неудивительно, что удаление ведет себя таким же образом.

**Итерирование.** Итерирование выполняется за  $O(N)$ , потому что для итерации по  $N$  элементам требуется  $N$  шагов. Это также объясняет, почему оператор `in`, `max`, `min` в Python является  $O(N)$ : чтобы определить, находится ли элемент в списке, мы должны перебирать каждый элемент.

**Срезы.** Чтобы получить доступ к фрагменту  $[a : b]$  списка, мы должны перебрать каждый элемент между индексами `a` и `b`. Таким образом, доступ к срезу –  $O(k)$ , где `k` – размер среза. Удаление среза  $O(N)$  по той же причине, что удаление одного элемента –  $O(N)$ : `N` последующих элементов должны быть смещены в сторону начала списка.

**Умножение на `int`.** Чтобы понять умножение списка на целое `k`, вспомним, что конкатенация выполняется за  $O(M)$ , где `M` – длина добавленного списка. Из этого следует, что умножение списка равно  $O(Nk)$ , так как умножение `k`-размера списка `N` раз потребует времени  $k(N - 1)$ .

**Разворот списка.** Разворот списка – это  $O(N)$ , так как мы должны переместить каждый элемент.

## Создание списков

Списки определяются с помощью квадратных скобочек или с помощью вызова конструктора `list()`. Можно создать список из одинаковых значений с помощью умножения, например `[0] * 10`. Чаще всего списки содержат переменные одного типа, например, строки. Однако бывает, что нужно добавлять в списки переменные разных типов. Также списки могут содержать другие коллекции (для таких данных чаще всего используются кортежи), как, например, `user_data`:

```
user_data = [
    ['Ivanov', 4.2],
    ['Romanov', 3.8]
]
```

**Пример.** Использование методов для работы со списками.

В [5]:

```
▼ # Использование методов для работы со списками
range_list = list(range(10))
print(range_list)
range_list[::-1]
```

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Out[5]:

[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

В [6]:

```
▼ # Сравнение списков
a = [1, 2, 3]
b = [1, 2, 3]
print(a == b)
b[0] = 5
print(a < b)
a[0] = [3, "aaa"]
b[0] = [3, "bbb"]
print(a, b)
a < b
```

True  
True  
[[3, 'aaa'], 2, 3] [[3, 'bbb'], 2, 3]

Out[6]:

True

В [7]:

```
▼ # Применение нескольких методов сортировки
ls = [-90, 78, 456, -890]
sorted(ls), ls
```

Out[7]:

([-890, -90, 78, 456], [-90, 78, 456, -890])

В [8]:

```
# Удалить элементы списка, расположенные между максимальным и минимальным элементами,
# включая сами максимум и минимум
# (предполагается, что в списке один максимальный и один минимальный элемент)
ls = [59, -70, -22, -49, -11, 100, 1, -38, -7, 60]
imx = ls.index(max(ls))
imn = ls.index(min(ls))
if imx < imn:
    del ls[imx : imn + 1]
else:
    del ls[imn : imx + 1]
ls
```

Out[8]:

```
[59, 1, -38, -7, 60]
```

Списки, как и все коллекции, **поддерживают протокол итерации**, значит, можно использовать цикл `for` для того, чтобы итерироваться по элементам списка. Итерации происходят именно по элементам списка, а не по индексам, как во многих языках.

В [9]:

```
# Вывод коллекции (вместе с индексами)
ls = [3, 66, 89, 34]
for i, item in enumerate(ls):
    print(i, '-', item)
```

```
0 - 3
1 - 66
2 - 89
3 - 34
```

## 4.2.2 Получение срезов из списка

Важно знать, что при получении среза **создается новый объект**, то есть в этот момент получается новый список. В действительности параметры среза определяются с помощью объекта среза, который и передается реализации операции индексирования списка. Фактически вы всегда можете передать объект среза вручную – синтаксис срезов в значительной степени является всего лишь **синтаксическим подсластителем** для операции индексирования с применением объекта среза.

# Извлечение среза с использованием синтаксиса срезов	# Извлечение среза с помощью объекта среза
<pre>&gt;&gt;&gt; L = [5, 6, 7, 8, 9] &gt;&gt;&gt; L[2:4] [7, 8] &gt;&gt;&gt; L[1:] [6, 7, 8, 9] &gt;&gt;&gt; L[:-1] [5, 6, 7, 8] &gt;&gt;&gt; L[::2] [5, 7, 9]</pre>	<pre>&gt;&gt;&gt; L = [5, 6, 7, 8, 9] &gt;&gt;&gt; L[slice(2, 4)] [7, 8] &gt;&gt;&gt; L[slice(1, None)] [6, 7, 8, 9] &gt;&gt;&gt; L[slice(None, -1)] [5, 6, 7, 8] &gt;&gt;&gt; L[slice(None, None, 2)] [5, 7, 9]</pre>

Пример. Получение срезов из списков.

В [10]:

```
# Получение срезов из списков
ls = [34, 67, 89, 3, 96, 5, 9, -63, 96, 12, -6]
ls[::-2]
```

Out[10]:

```
[34, 89, 96, 9, 96, -6]
```

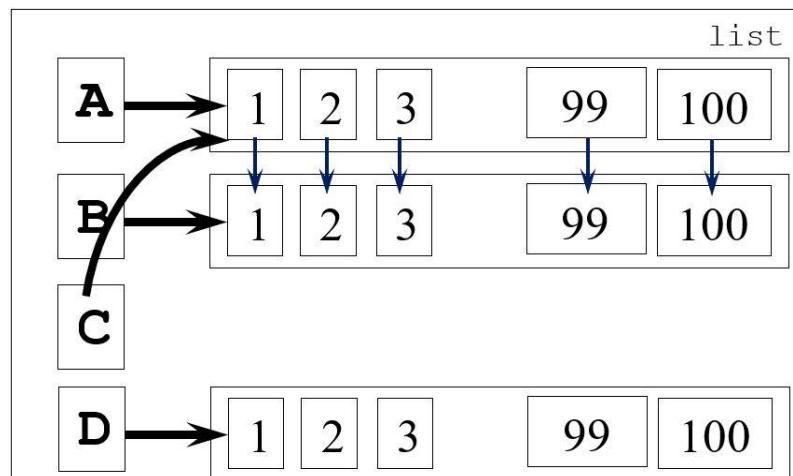
#### 4.2.3 Копирование списков

Посмотрим, как происходит работа со списками в случае их копирования при выполнении фрагмента кода (можно поэкспериментировать на сайте [pythontutor.com \(http://pythontutor.com/\)](http://pythontutor.com/)):

```
N = int(input())
A = [0] * N
B = [0] * N
for k in range(N):
    A[k] = int(input())

# Поэлементное копирование
for k in range(N):
    B[k] = A[k]

C = A    # Копируется ссылка
# Создание списка- дубликата A
D = list(A)
```



В [ ]:

```
N = int(input())
A = [0] * N
B = [0] * N
for k in range(N):
    A[k] = int(input())

# Поэлементное копирование
for k in range(N):
    B[k] = A[k]

C = A # Копируется ссылка
# Создание списка- дубликата A
D = list(A)
```

#### 4.2.4 Списковые включения (генерация списков)

Списковые включения в Python представляют собой **способ компактного описания порождения списка**. Нотация вдохновлена математической записью для построения множества, а также языком программирования SETL.

В математике множество квадратов чисел от 1 до 9 можно определить, как

$$\{n^2 : \forall n \in N, 1 \leq n < 10\}$$

В Python список, который содержит приведенное выше множество чисел, можно описать следующим образом:

В [12]:

```
[n ** 2 for n in range(1, 10)]
```

Out[12]:

```
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

При генерации списка можно указать дополнительное условие, как показано в примере ниже:

В [13]:

```
# Список четных чисел в заданном диапазоне
[n for n in range(1, 10) if n % 2 == 0]
```

Out[13]:

```
[2, 4, 6, 8]
```

Упрощенный шаблон спискового включения выглядит так:

[выражение **for** переменная **in** последовательность **if** условие]

Это соответствует следующему псевдокоду:

```
результат = []
for переменная in последовательность:
    if условие:
        результат.append(выражение)
```

Реальный синтаксис спискового включения сложнее, поскольку допускает вложенность, которой иногда программисты злоупотребляют.

**Примеры** генерации списков.

В [14]:

```
# Список из 10 случайных целых чисел от -100 до 100
from random import randint

[randint(-100, 100) for _ in range(10)]
```

Out[14]:

```
[-75, -88, -4, 69, -6, 100, -40, -73, -33, 29]
```

В [15]:

```
# Список квадратов чисел от 0 до 20, кратных 3
[x**2 for x in range(21) if x % 3 == 0]
```

Out[15]:

```
[0, 9, 36, 81, 144, 225, 324]
```

В [16]:

```
# Определить порядковые номера максимальных элементов списка
ls = [59, -70, -22, -49, -11, 100, 1, -38, -7, 60, 100, -100, 100, -90]
[i + 1 for i, j in enumerate(ls) if j == max(ls)]
```

Out[16]:

```
[6, 11, 13]
```

## 4.2.5 Вложенные списки

В Python **отсутствует встроенный тип многомерного массива**. Тем не менее, имитировать многомерный массив можно с помощью обычных списков, как показано в примере ниже.

В [17]:

```
arr = [[x * y for x in range(5)] for y in range(3)]
arr
for y in range(3):
    for x in range(5):
        print(f'arr[{y}][{x}]={arr[y][x]}', end=' ')
print()
```

```
arr[0,0]=0 arr[0,1]=0 arr[0,2]=0 arr[0,3]=0 arr[0,4]=0
arr[1,0]=0 arr[1,1]=1 arr[1,2]=2 arr[1,3]=3 arr[1,4]=4
arr[2,0]=0 arr[2,1]=2 arr[2,2]=4 arr[2,3]=6 arr[2,4]=8
```

**Пример.** Пример инициализации матрицы формата  $N$  на  $M$ .

В [18]:

```
# Пример инициализации матрицы формата N на M
n = 8
m = 3
mass = []
for i in range(n):
    mass.append([0] * m)
print(mass)
```

```
[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

**Пример** генерации и вывода матрицы формата  $N$  на  $M$ .

В [19]:

```
# Пример генерации и вывода матрицы формата N на M
from random import randint

n, m = map(int, input('Число строк и столбцов: ').split())
a = []
for i in range(n):
    a.append([0] * m)
for i in range(n):
    for j in range(m):
        a[i][j] = randint(0, 100)
        print("%4d" % (a[i][j]), end=' ')
    print()
```

Число строк и столбцов: 2 3

```
89   70   58
41   19   73
```

**Пример** ручного ввода и вывода матрицы формата  $N$  на  $M$ .

В [21]:

```
# Пример ручного ввода и вывода матрицы формата N на M.
n, m = map(int, input('Число строк и столбцов: ').split())
a = []
for i in range(n):
    a.append([0] * m)

for i in range(n):
    for j in range(m):
        print("a[{0}][{1}] = ".format(i, j), end="")
        a[i][j] = int(input())

print('\nИсходная матрица:')
for i in range(n):
    for j in range(m):
        print("%4d" % (a[i][j]), end=' ')
    print()
```

Число строк и столбцов: 2 3

```
a[0][0] = -6
a[0][1] = -5
a[0][2] = -63
a[1][0] = 12
a[1][1] = 35
a[1][2] = -8
```

Исходная матрица:

```
-6   -5   -63
12   35   -8
```

В [22]:

```
# Бонусное задание
import numpy as np

price = [1_000_000, 10_000, 1_000_000, 10_000, 10_000, 10_000]
print(np.mean(price))
print(np.median(price))
```

```
340000.0
10000.0
```

## 4.2.6 Примеры решения задач

**Задача 1.** В произвольно заданном одномерном массиве определить число отрицательных, нулевых и положительных элементов. Решить задачу двумя способами (анализом каждого элемента и с использованием генераторов списков) и сравнить время работы.

```

from random import randint
from datetime import datetime

n = int(input('Число элементов = '))
ls = [randint(-100, 100) for _ in range(n)]
# print('Сгенерированный список: \n', ls)

start_time = datetime.now()
knegative = 0
kzero = 0
kpositiv = 0
for a in ls:
    if a < 0:
        knegative += 1
    elif a == 0:
        kzero += 1
    else:
        kpositiv += 1
print('Число отрицательных элементов:', knegative)
print('Число нулевых элементов:', kzero)
print('Число положительных элементов:', kpositiv)
print('Время работы %s.\n' % (datetime.now() - start_time))

start_time = datetime.now()
print('Число отрицательных элементов:', len([x for x in ls if x < 0]))
print('Число нулевых элементов:', len([x for x in ls if x == 0]))
print('Число положительных элементов:', len([x for x in ls if x > 0]))
print('Время работы %s.' % (datetime.now() - start_time))

```

Число элементов = 20000000  
 Число отрицательных элементов: 9948905  
 Число нулевых элементов: 99032  
 Число положительных элементов: 9952063  
 Время работы 0:00:08.722864.

Число отрицательных элементов: 9948905  
 Число нулевых элементов: 99032  
 Число положительных элементов: 9952063  
 Время работы 0:00:08.770153.

**Задача 2.** В произвольно заданном одномерном массиве определить минимальный и максимальный элементы и поменять их местами.

В [25]:

```
from random import randint

n = int(input('Число элементов = '))
ls = [randint(-100, 100) for _ in range(n)]
print('Сгенерированный список: \n', ls)

mn = min(ls)
i_mn = ls.index(mn)
mx = max(ls)
i_mx = ls.index(mx)
print('Минимальный элемент:', mn)
print('Максимальный элемент:', mx)

ls[i_mn] = mx
ls[i_mx] = mn
print(ls)
```

Число элементов = 10  
Сгенерированный список:  
[-85, -100, -15, 34, -33, -95, 80, 46, 10, -97]  
Минимальный элемент: -100  
Максимальный элемент: 80  
[-85, 80, -15, 34, -33, -95, -100, 46, 10, -97]

**Задача 3.** Определить в списке два элемента с наибольшими значениями.

В [26]:

```
from random import randint

n = int(input('Число элементов = '))
ls = [randint(-100, 100) for _ in range(n)]
print('Сгенерированный список: \n', ls)

mx = sorted(ls)[-2:]
print('Второй максимум = %d. Первый максимум = %d.' % (mx[0], mx[1]))
```

Число элементов = 10  
Сгенерированный список:  
[97, -28, -44, 56, 60, 35, 59, -97, -52, 70]  
Второй максимум = 70. Первый максимум = 97.

**Задача 4.** Изменить порядок расположения элементов в списке, расположив сначала отрицательные элементы исходного списка, потом нулевые и в последнюю очередь – положительные, сохранив порядок следования элементов.

В [27]:

```
from random import randint

n = int(input('Число элементов = '))
ls = [randint(-100, 100) for _ in range(n)]
print('Сгенерированный список: \n', ls)

ls = [x for x in ls if x < 0] + [x for x in ls if x == 0] + [x for x in ls if x > 0]
print('Измененный список: \n', ls)
```

Число элементов = 20

Сгенерированный список:

[90, 28, -35, -68, -60, 47, 12, -51, -76, 30, 56, 39, -76, 36, 91, 95, 23, 66, -29, 81]

Измененный список:

[-35, -68, -60, -51, -76, -76, -29, 90, 28, 47, 12, 30, 56, 39, 36, 91, 95, 23, 66, 81]

**Задача 5.** Дан список целых чисел. Удалить из него все одинаковые элементы, оставив их последние вхождения.

В [28]:

```
from random import randint

n = int(input('Число элементов = '))
ls = [randint(-5, 5) for _ in range(n)]
print('Сгенерированный список: \n', ls)

i = 0
while i < len(ls):
    if ls[i] in ls[i+1:]:
        del ls[i]
    else:
        i += 1
print('Измененный список: \n', ls)
```

Число элементов = 20

Сгенерированный список:

[3, -3, -4, 1, -2, -3, -5, -5, 3, -1, -2, -3, -1, -1, 1, 4, 0, 2, 2, -3]

Измененный список:

[-4, -5, 3, -2, -1, 1, 4, 0, 2, -3]

## 4.2.7 Контрольное задание. Использование списков

**Задание.** Найти медиану случайного списка.

**Медиана** – это значение в отсортированном списке, которое лежит ровно посередине, таким образом, половина значений – слева от него, и половина значений – справа. Если у нас количество элементов в списке нечётное, то просто берём средний элемент. Если количество элементов чётное, то по определению медианы нам нужно взять среднее арифметическое от двух средних элементов.

В [29]:

```
import random

numbers = []
numbers_size = random.randint(10, 25)

for _ in range(numbers_size):
    numbers.append(random.randint(100, 200))
print(numbers)
numbers.sort()

half_size = len(numbers) // 2
median = None
if numbers_size % 2 == 1:
    median = numbers[half_size]
else:
    median = sum(numbers[half_size - 1:half_size + 1]) / 2
print(median)
```

```
[142, 127, 107, 162, 173, 188, 148, 127, 167, 166, 160, 182, 193, 174, 160,
181, 196, 183, 161, 113, 147, 194, 158, 173, 155]
```

162

Чтобы это проверить правильность вычисления медианы, можно воспользоваться встроенным модулем `statistics`, который найдет медиану намного быстрее. Импортируем модуль `statistics`, и воспользуемся методом `median`, который позволит найти медиану.

В [30]:

```
import statistics
print(statistics.median(numbers))
```

162

#### 4.2.8 Задачи для самостоятельного выполнения

Во всех задачах необходимо обработать два списка:  $A$  – сгенерированный случайным образом список целых чисел,  $B$  – введенный с клавиатуры список действительных чисел.

**Задача 1.** Поменять местами максимальные и минимальные элементы в списке  $B$ . В списке  $A$  поменять порядок следования элементов, расположив сначала положительные, потом отрицательные элементы. Нулевые элементы исключить из списка  $A$ .

**Задача 2.** Поменяйте местами элементы списков, стоящие на четных позициях (т.е. должны поменяться 2, 4, 6 и т.д. элементы местами). Списки могут быть разного размера. Если какой-то список длиннее, то его "хвост" останется неизмененным.

**Задача 3.** Привести все элементы из  $B$  к целым значениям, отбросив дробную часть (т.е. без округления). Определить сколько значений из списка  $A$  встречаются в списке  $B$ . Обнулить элементы, присутствующие в обоих списках.

**Задача 4.** В списке  $B$  поменять местами первый элемент с минимальным, а последний – с максимальным. В списке  $A$  поменять местами два соседних элемента местами: первый со вторым, третий – с четвертым и т.д.

**Задача 5.** Переставить в обратном порядке элементы списка  $B$ , расположенные между его минимальным и максимальным элементами, включая минимальный и максимальный элементы. В списке  $A$  все элементы нормировать: разделить их на сумму всех элементов. Проверить, что после нормировки сумма всех элементов в списке  $A$  будет равна 1.

**Задача 6.** Найти количество участков, на которых элементы списка  $B$  монотонно убывают. В списке  $A$  найти количество "впадин" – число элементов (крайние не рассматриваются), у которых "соседи" элемента строго больше.

**Задача 7.** Все элементы списка  $A$  увеличить на число, равное индексу этого элемента (первый элемент – имеет индекс ноль, он не изменится, второй – с индексом один увеличить на один и т.д.). Проверить, образуют ли элементы списка  $B$  геометрическую прогрессию.

**Задача 8.** В списке  $A$  обнулить элементы, расположенные между двумя одинаковыми (предполагается, что в списке будет хотя бы одна пара одинаковых элементов). Если имеется несколько пар одинаковых элементов, то обнулить так элементы, чтобы длина последовательности подряд идущих нулей была максимальна. Проверить, можно ли из элементов списка  $B$  составить арифметическую прогрессию.

**Задача 9.** В списке  $A$  удалить элементы, расположенные между двумя одинаковыми (предполагается, что в списке будет хотя бы одна пара одинаковых элементов). Если имеется несколько пар одинаковых элементов, то удалить так элементы, чтобы удаляемая часть списка была минимальной. В списке  $B$  удалить элементы с четных позиций, которые оказались по модулю больше 10.

**Задача 10.** В списке  $A$  оставить только простые числа. Расширить список  $B$ , добавляя в конец удаляемые из списка  $A$  элементы. Определить среднее арифметическое значений элементов получившегося списка  $B$ , стоящих на четных позициях.

## § 4.3 Кортежи

Кортеж (`tuple`) – это упорядоченная **неизменяемая последовательность** элементов.

**Особенности:** умеет все, что умеет список, за исключением операций, приводящих к изменению кортежа. Применяется в случаях, когда известно, что **последовательность не будет меняться** после создания – мы не можем ни добавлять, ни удалять элементы из него. Кортежи определяются с помощью круглых скобочек или конструктора `tuple`. Кортежи поддерживают все операции, общие для последовательностей.

**Важно:** при определении кортежа **из одного элемента**, не забывайте писать запятую, потому что если вы забудете про нее, то Python сочтет вашу переменную типом `int`.

**Примеры** создания кортежа.

В [31]:

```
# Пустой кортеж
tp = ()

# Кортеж из одного элемента,
# запятая нужна, чтобы отличать от выражения в скобках
tp1 = ('single',)

# Еще один кортеж
tp2 = (1, 2, ["three", 4])
# Скобки можно опустить, важны запятые
tp3 = 1, 2
print(tp, tp1, tp2, tp3)
```

() ('single',) (1, 2, ['three', 4]) (1, 2)

**Замечание.** Кортежам не нужно заботиться о том, что им придётся меняться, поэтому они занимают меньше места и часто их работа может быть оптимизирована интерпретатором. Важно понимать, что эти различия настолько незаметны, что играют роль только в тех местах, где борьба действительно идёт за байты и наносекунды. Так что **смело используйте списки**. Несмотря на то, что сами кортежи неизменяемые, объекты внутри них могут быть изменяемыми. Например, если у нас кортеж содержит список, мы можем добавлять элементы в этот список. Например, мы можем присоединить нолик.

В [ ]:

```
etalon = (30000, 2000, 3, 1)
vak1 = (10000000, 100, 0, 0)
```

В [32]:

```
# Пример изменения элемента кортежа
a = [5, 6]
tp = (1, 2, a, 4)
a[0] = 'z'
tp
```

Out[32]:

(1, 2, ['z', 6], 4)

В [33]:

```
a, b = 10, 20
print(a,b)
```

10 20

С помощью кортежей можно организовать **множественное присваивание значений переменных**:

В [34]:

```
data = ['Python', ('strong', 'dynamic'), 30]
language, (ty1, ty2), value = data
print(language, value)
ty1, ty2 = ty2, ty1
print(ty1, ty2)
```

```
Python 30
dynamic strong
```

Множественное присваивание удобно использовать в цикле `for` с итератором `enumerate()`, позволяющим получить как индекс, так и значение для элементов последовательности:

В [35]:

```
for i, item in enumerate('abcdef'):
    print(i, item, sep=':', end=' ')
```

```
0:a 1:b 2:c 3:d 4:e 5:f
```

Также важная особенность кортежей – у них есть функция `hash` и поэтому они могут использоваться в качестве ключей в словарях (чуть позднее).

В [36]:

```
hash(tuple())
```

Out[36]:

```
5740354900026072187
```

## § 4.4 Числовой диапазон (range)

**Числовой диапазон ( range )** – это упорядоченная **неизменяемая последовательность элементов** – целых чисел.

Конструктор класса - `range`. Параметры:

- `start (int)` – начальное значение (по умолчанию 0);
- `stop (int)` – конечное значение (не включается в результат);
- `step (int)` – шаг изменения (по умолчанию 1, может быть отрицательным).

Числовые диапазоны поддерживают те же операции, что и кортежи.

**Пример.** Создание числового диапазона

В [37]:

```
# Создание числового диапазона
list(range(1, -10, -2))
```

Out[37]:

```
[1, -1, -3, -5, -7, -9]
```

## § 4.5 Множества

### 4.5.1 Определение множества и его создание

Тип **множество** понимается в математическом смысле — как **неупорядоченных коллекция уникальных** элементов. Элементы могут быть **различных типов**. Порядок элементов не определён. Неупорядоченность элементов в множестве позволяет быстро выполнять операции над множествами.

Множество, аналогично словарю, задается перечислением в фигурных скобках, но без указания ключей:

```
>>> A = {1, 3, 5}
```

Исключением является **пустое множество**, которое **создается с помощью функции set()**.

```
>>> B = set()      # B – пустое множество
>>> D = {}        # D – не пустое множество, а пустой словарь!
```

Многие операции имеют названия, отличные от общих для типов-последовательностей:

В [38]:

```
firms = {'Yandex', 'Yandex', 'JetBrains', 'MCST'}
set(['Yandex', 'JetBrains', 'MCST']) # еще один вариант определения множества
firms.add('Kaspersky')
firms.remove('JetBrains')
print(firms, 'Kaspersky' in firms, {x for x in firms if 'a' in x})
```

```
{'Yandex', 'MCST', 'Kaspersky'} True {'Yandex', 'Kaspersky'}
```

В Python существует 2 класса для работы с множествами:

- `class set([iterable])` – конструктор класса `set` – изменяемое множество;
- `class frozenset([iterable])` – конструктор класса `frozenset` – неизменяемое множество.

Оба типа обладают различиями, схожими с различиями между списком и кортежем.

Наиболее часто множества **используются** для эффективной проверки на вхождение, удаления повторяющихся элементов, а также выполнения математических операций, характерных для математических множеств (пересечение, объединение и др.).

Действия, которые можно выполнять с множеством:

- добавлять и удалять элементы,
- проверять принадлежность элемента множеству,

- перебирать его элементы,
- выполнять операции над множествами (объединение, пересечение, разность).

Операция “проверить принадлежность элемента” выполняется в множестве намного быстрее, чем в списке. Элементами множества может быть любой неизменяемый тип данных: числа, строки, кортежи (элементы множества должны быть хешируемы). Изменяемые типы данных не могут быть элементами множества, в частности, нельзя сделать элементом множества список (вместо этого используйте неизменяемый кортеж) или другое множество. Требование неизменяемости элементов множества накладывается особенностями представления множества в памяти компьютера.

Следует обратить внимание:

- множество может содержать элементы разных типов;
- т.к. порядок элементов в множестве не определен (множество – неупорядоченный набор данных), то при выводе порядок его элементов может быть произвольным (Python выводит элементы множества в случайном порядке);
- если функции `set` передать в качестве параметра список, строку или кортеж, то она вернет множество, составленное из элементов списка, строки, кортежа;
- каждый элемент множества уникален (входит в множество один раз).

Python гарантирует то, что в множестве содержатся уникальные элементы. Это достигается с помощью функции хеширования. Поскольку для хранения объектов во множестве используется функция хеширования, то в множестве могут содержаться неизменяемые объекты. Неизменяемость гарантирует, что хэш от объекта останется прежним, и мы сможем найти нужный адрес памяти в множестве при поиске элемента.

В [31]:

```
animals_and_digit = {7, 'rabbit', 5, 'goat', 7, 'cow', 8, 'dog', 3, 'cat', 7}
print(animals_and_digit)
```

{3, 5, 7, 8, 'cow', 'dog', 'cat', 'rabbit', 'goat'}

В [39]:

```
# Из множества можно сделать список при помощи функции list
mn = {1, 5, 5, 5, 6, 7, 1, 7}
ls = list(mn)
ls
```

Out[39]:

[1, 5, 6, 7]

В [40]:

```
# При помощи цикла for можно перебрать все элементы множества
mn = {'Yandex', 'JetBrains', 'MCST', 'Kaspersky'}
for item in mn:
    print(item)
```

Yandex  
MCST  
JetBrains  
Kaspersky

## 4.5.2 Операции над множествами

### 4.5.2.1 Операции, общие для последовательностей

Множества поддерживают все общие операции для последовательностей (такие как `in`, `not in`, `len`, `max`, `min`), и имеют ряд дополнительных методов.

Очень часто требуется **обойти все элементы множества**. Для этого используется цикл `for` и оператор `in`.

**Пример.** Определить сумму чисел, входящих в множество.

В [42]:

```
animals_and_number = {11, 'rabbit', 5.6, 'goat', 7, 'cow', 8, 'dog', 3, 'cat', 12.5}
sm = 0
for elem in animals_and_number:
    if isinstance(elem, (int, float)):
        sm += elem
print(sm)
```

47.1

Операция `in` используется для проверки наличия элемента в множестве. Интересен тот факт, что эта операция для множеств в Python выполняется за время, не зависящее от мощности множества (количество его элементов).

**Пример.** Проверить вхождение некоторого элемента в множество.

В [43]:

```
my_set = {'rabbit', 'goat', 'cow', 'dog', 'cat'}
elem = 'fox'
if elem in my_set:
    print('Элемент входит в множество')
else:
    print('Элемент не входит в множество')
```

Элемент не входит в множество

Если множество содержит элементы одного типа, то для него можно использовать функции нахождения максимального и минимального элементов. Для предыдущего примера использование функции `max()` или `min()` приведет к возникновению исключения `TypeError`.

**Пример.** Поиск максимального и минимального элемента в множестве из элементов одного типа.

В [38]:

```
animals = {'rabbit', 'goat', 'cow', 'dog', 'cat'}
number = {11, -5, 7, 8, 32, 2}
print(max(animals))
print(min(number))
```

```
rabbit
-5
```

#### 4.5.2.2 Операции над множествами, приводящие к его изменению

Рассмотрим операции, приводящие к изменению объекта, доступные только для множеств `set`.

Таблица. Общие методы для множеств ( А – множество, на котором вызывается метод)

Синтаксис метода	Описание метода	Трудоемкость
<code>A.add(elem)</code>	Добавляем элемент <code>elem</code> в множество <code>st</code> . Если <code>elem</code> в множестве уже существует, то оно не изменится (множество не может содержать одинаковых элементов)	$O(1)$
<code>A.remove(elem)</code>	Удаляет элемент <code>elem</code> из множества <code>st</code> . Если элемент не находится в множестве, возникает ошибка <code>KeyError</code>	$O(1)$
<code>A.discard(elem)</code>	Удаляет элемент <code>elem</code> из множества <code>st</code> , если он присутствует в множестве, если элемента в множестве нет - ничего не делает	$O(1)$
<code>A.pop()</code>	Удаляет случайный элемент из множества <code>st</code> и возвращает в качестве результата. Применение метода к пустому множеству приведет к ошибке <code>KeyError</code>	$O(1)$
<code>A.clear()</code>	Удаляет все элементы из множества	$O(1)$

Как мы видим, по времени стандартные операции с одним элементом множества выполняются за  $O(1)$ . В случае, если нужно провести процедуру, затрагивающую все элементы множества, то его трудоемкость будет  $O(N)$ .

Пример. Добавление элемента в множество.

В [44]:

```
my_set = {'rabbit', 'goat', 'cow', 'dog', 'cat'}
elem = 'fox'
my_set.add(elem)
print(my_set)
```

```
{'cow', 'dog', 'fox', 'cat', 'rabbit', 'goat'}
```

Как видно из таблицы для удаления одного элемента из множества существует сразу три метода: `discard`, `remove`, `pop`.

Необходимо понимать, что:

1. метод `pop` удаляет из множества **случайный элемент**, который при этом возвращается методом, т.е. его можно посмотреть;
2. метод `remove` **может породить ошибку** в случае попытки удаления не существующего в множестве элемента, поэтому при его использовании лучше предварительно проверить (используя

in ) факт наличия удаляемого элемента в множестве.

**Пример.** Удаление элементов из множества.

B [45]:

```
my_set = {'cow', 'dog', 'fox', 'cat', 'rabbit', 'goat'}  
  
my_set.discard('fox')      # Удален  
my_set.discard('wolf')    # Не удален, но ошибки нет  
print(my_set)  
del_elem = my_set.pop()   # Удален случайный элемент  
print('Удален:', del_elem)  
print(my_set)  
my_set.remove('elk')      # Не удален, ошибка KeyError
```

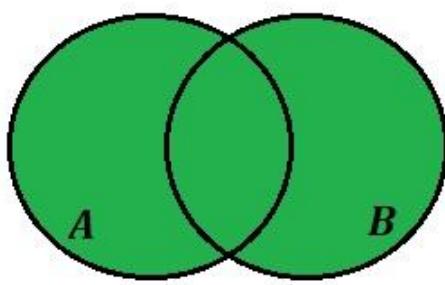
```
{'cow', 'dog', 'cat', 'rabbit', 'goat'}  
Удален: cow  
{'dog', 'cat', 'rabbit', 'goat'}
```

```
-----  
KeyError                                                 Traceback (most recent call last)  
<ipython-input-45-f44fd22b8eff> in <module>  
      7 print('Удален:', del_elem)  
      8 print(my_set)  
----> 9 my_set.remove('elk')  
  
KeyError: 'elk'
```

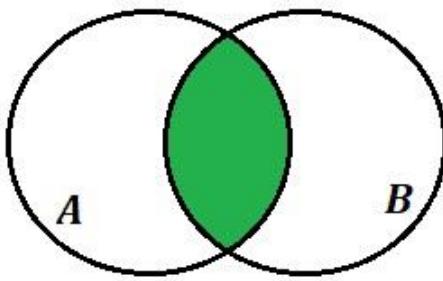
#### 4.5.2.3 Математические операции над множествами

Множества **поддерживают математические операции**, характерные для множеств (пересечение, объединение и др.).

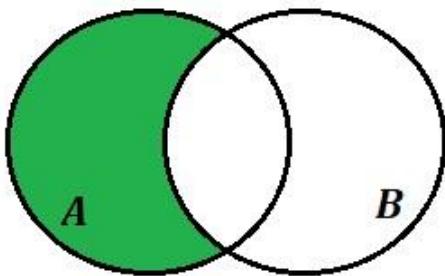
*Объединение  $A \cup B$*



*Пересечение  $A \cap B$*



*Разность  $A \setminus B$*



*Симметрическая разность  $A \Delta B$*

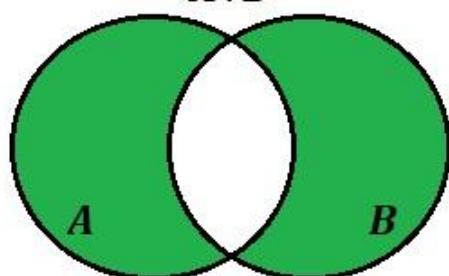


Таблица. Математические операции для множеств ( А – множество, на котором вызывается метод)

Синтаксис метода	Описание метода	Трудоемкость
<code>A.union(B, ...), A   B   ...</code>	Возвращает новое множество – объединение множеств А и В	$O(\text{len}(A)+\text{len}(B))$
<code>A.update(B, ...), A  = B   ...</code>	Записывает в А объединение множеств А и В (элементы, которые есть хотя бы в одном множестве)	$O(\text{len}(A)+\text{len}(B))$
<code>A.intersection(B, ...), A &amp; B &amp; ...</code>	Возвращает новое множество – пересечение множеств А и В (элементы, которые есть в обоих множествах)	$O(\min(\text{len}(A), \text{len}(B)))$
<code>A.difference(B, ...), A - B - ...</code>	Возвращает новое множество – разность множеств А и В (элементы, входящие в А , но не входящие в В )	$O(\text{len}(A)+\text{len}(B))$
<code>A.symmetric_difference(B), A ^ B</code>	Возвращает симметрическую разность множеств А и В (элементы, входящие в А или в В , но не в оба из них одновременно)	$O(\text{len}(A)+\text{len}(B))$
<code>A.isdisjoint(B)</code>	Возвращает True если А не содержит общих элементов с В	
<code>A.issubset(B), A &lt;= B</code>	Возвращает True если все элементы А содержатся в В ( А – подмножество В )	$O(\text{len}(A))$
<code>A &lt; B</code>	Аналогично <code>A &lt;= B</code> , но множества не должны полностью совпадать ( <code>A &lt;= B</code> and <code>A != B</code> )	$O(\text{len}(A))$
<code>A.issuperset(B), A &gt;= B</code>	Возвращает True если все элементы В содержатся в А ( В – подмножество А )	$O(\text{len}(B))$
<code>A &gt; B</code>	Аналогично <code>A &gt;= B</code> , но множества не должны полностью совпадать	$O(\text{len}(B))$

Пример. Даны четыре множества:

```
A = set('0123456789')
B = set('02468')
C = set('12345')
D = set('56789')
```

Найти элементы, принадлежащие множеству

$$((A \setminus B) \cap (C \setminus D)) \cup ((D \setminus A) \cap (B \setminus C))$$

В [43]:

```
# Пример работы с множеством
A = set('0123456789')
B = set('02468')
C = set('12345')
D = set('56789')
((A - B) & (C - D)) | ((D - A) & (B - C))
```

Out[43]:

```
{'1', '3'}
```

#### 4.5.2.4 Сравнение множеств

Все операторы сравнения множеств, а именно `==`, `<`, `>`, `<=`, `>=`, `!=` возвращают `True`, если сравнение истинно, и `False` – в противном случае.

Множества считаются **равными**, если они содержат одинаковые наборы элементов.

Равенство множеств проверяется оператором `==`, неравенство оператором `!=`.

В [1]:

```
set_1 = {4, 6, 2, 8}
set_2 = {2, 4, 6, 8, 2}
if set_1 == set_2:
    print('Множества равны')
else:
    print('Множества не равны')
```

Множества равны

#### 4.5.2.5 Подмножество и надмножество

**Подмножество** – некоторая выборка элементов множества, которая может быть как меньше множества, так и совпадать с ним. **Надмножество** может включать **все** элементы некоторого множества и, возможно, какие-то еще.

Операция  $A \subset B$  означает, что  $A$  является подмножеством  $B$ , но целиком не совпадает с ним. В Python подобного рода сравнение запишется: `A < B`.

Операция  $A \subseteq B$  означает, что  $A$  является подмножеством  $B$ , или совпадает с ним. В Python подобного рода сравнение запишется: `A <= B`.

Операция  $A \supseteq B$  означает, что  $A$  является надмножеством  $B$ , или совпадает с ним. В Python подобного рода сравнение записывается:  $A \geq B$ .

В [5]:

```
s1 = {'a', 'b', 'c', 'd'}
s2 = {'a', 'b', 'd'}
s3 = {'c', 'z'}

print(s1 <= s1)      # True
print(s2 < s1)       # True
print(s3 <= s1)      # False
```

True  
True  
False

#### 4.5.3 Неизменяемое множество frozenset

Неизменяемое множество `frozenset` гарантирует уникальность элементов и ведет себя точно так же, как и множества, но мы **не можем добавлять туда элементы или удалять их**. Например, мы можем определить множество `frozen` и попытаться в него что-то добавить. У нас ничего не выйдет, потому что `frozenset` неизменяемый, у него нет атрибута `add`.

В [44]:

```
frozen = frozenset(['Нюша', 'Бараш', 'Крош'])
frozen.add('Ёжик')
```

```
-----  
AttributeError                                     Traceback (most recent call last)
<ipython-input-44-28d60e434470> in <module>
      1 frozen = frozenset(['Нюша', 'Бараш', 'Крош'])
----> 2 frozen.add('Ёжик')

AttributeError: 'frozenset' object has no attribute 'add'
```

#### 4.5.4 Примеры решения задач

**Задача.** Для случайно сгенерированного списка целых чисел определить, есть ли в нем одинаковые элементы.

В [55]:

```
from random import randint

n = int(input('Число элементов = '))
ls = [randint(-100, 100) for _ in range(n)]
print('Сгенерированный список: \n', ls)

if len(ls) == len(set(ls)):
    print('В списке все элементы уникальны')
else:
    print('В списке есть одинаковые элементы')
```

Число элементов = 10  
Сгенерированный список:  
[75, 60, 49, -53, -56, 44, 29, -71, -6, 75]  
В списке есть одинаковые элементы

**Задача (однофамильцы).** Начальник кадровой службы хочет узнать, сколько однофамильцев-мужчин работает в организации. у него есть список фамилий на основании которого нужно определить количество фамилий, которые совпадают с другими. При вводе в первой строке указывается  $N$  – количество сотрудников организации, затем идут  $N$  строк с фамилиями этих сотрудников в произвольном порядке.

### Тестовый пример

Ввод:

```
6
Иванов
Петров
Сидоров
Петров
Иванов
Петров
```

Выход:

5

```

n = int(input())
workers = []
original = set()
for _ in range(n):
    family = input()
if family not in workers and family not in original:
    original.add(family)
else:
    original.discard(family)
workers.append(family)
print(n - len(original))

```

6  
Иванов  
Петров  
Сидоров  
Петров  
Иванов  
Петров  
5

**Задача (прощальный ужин).** Повар курортного отеля решил в последний вечер перед отъездом гостей их порадовать: приготовить блюда, которые он им еще не готовил из своего рецептурного меню. В его распоряжении есть список из  $M$  блюд его рецептурного меню,  $N$  - число дней, для которых есть списки блюд (которые он готовил для гостей), далее по каждому дню есть количество приготовленных блюд и их название. Составьте список блюд, которые еще не готовил повар.

### Тестовый пример

*Ввод:*

5  
Овсянка  
Плов  
Борщ  
Гречка с подливой  
Рыба  
2  
3  
Плов  
Борщ  
Рыба  
2  
Плов  
Рыба

*Выход:*

Гречка с подливой  
Овсянка

В [2]:

```
M = int(input('Сколько блюд может приготовить повар: '))
menu = set()
for i in range(M):
    menu.add(input(f' Введите название {i+1} блюда: '))

N = int(input('Число дней, для которых есть списки блюд: '))
was_prepared = set()
for d in range(N):
    k = int(input(f' Сколько блюд готовил повар в {d + 1} день: '))
    for _ in range(k):
        was_prepared.add(input(' Название блюда: '))

original_menu = menu - was_prepared
print('Список блюд, которые повар еще ни разу не готовил:')
for food in original_menu:
    print(' ', food)
```

```
Сколько блюд может приготовить повар: 5
Введите название 1 блюда: Овсянка
Введите название 2 блюда: Плов
Введите название 3 блюда: Борщ
Введите название 4 блюда: Гречка с подливой
Введите название 5 блюда: Рыба
Число дней, для которых есть списки блюд: 2
Сколько блюд готовил повар в 1 день: 3
Название блюда: Плоа
Название блюда: Борщ
Название блюда: Рыба
Сколько блюд готовил повар в 2 день: 2
Название блюда: Плов
Название блюда: Рыба
Список блюд, которые повар еще ни разу не готовил:
Овсянка
Гречка с подливой
```

**Задача (в ожидании гостей).** Неожиданно в отель после отъезда постояльцев, которых повар хорошо накормил, заехали раньше времени новые постояльцы. С дороги они были очень голодны. Повару надо их поразить своим кулинарным мастерством, но времени на закупку продуктов у него нет. Повару придется довольствоваться теми продуктами, которые остались у него в холодильнике. В кулинарной книге повара для каждого блюда перечислены необходимые ингредиенты. Определите какие блюда из своей рецептурной книги повар может приготовить. Считаем, что если продукт содержится в холодильнике, то его достаточно для приготовления блюда по любому рецепту.

Во входных данных в первой строке указывается число продуктов в холодильнике  $M$ . Далее идут  $M$  строчек с названиями продуктов. После этого – строчка с числом рецептов  $N$ . Далее –  $N$  блоков, описывающих каждый из рецептов. Блок начинается со строчки с названием рецепта, затем, на следующей строке – количество ингредиентов в нем. Далее идут названия ингредиентов, состоящие из одного слова.

На выходе должен получиться перечень из блюд, которые можно приготовить из продуктов в холодильнике, в порядке их появления в книге рецептов.

**Тестовый пример**

*Ввод:*

4	
Яблоки	
Хлеб	
Варенье	
Картошка	
3	
Тосты	
2	
Хлеб	
Варенье	
Яблочный сок	
1	
Яблоки	
Яичница	
1	
Яйца	

*Выход:*

Тосты	
Яблочный сок	

В [5]:

```
M = int(input('Сколько продуктов в холодильнике: '))
ingredients = set()
for i in range(M):
    ingredients.add(input(f' Название {i+1} продукта из холодильника: '))

menu= []
N = int(input('Число рецептов: '))
for i in range(N):
    name_food = input(f'Название {i + 1} рецепта: ')
    food = set()
    k = int(input(f' Сколько ингридиентов в {name_food}: '))
    for _ in range(k):
        food.add(input(' Название ингридиента: '))
    if food <= ingredients:
        menu.append(name_food)

print('Список рецептов, которыми можно порадовать гостей:')
for food in menu:
    print(' ', food)
```

Сколько продуктов в холодильнике: 4  
Название 1 продукта из холодильника: Яблоки  
Название 2 продукта из холодильника: Хлеб  
Название 3 продукта из холодильника: Варенье  
Название 4 продукта из холодильника: Картошка  
Число рецептов: 3  
Название 1 рецепта: тосты  
Сколько ингридиентов в тосты: 2  
Название ингридиента: Хлеб  
Название ингридиента: Варенье  
Название 2 рецепта: яблочный сок  
Сколько ингридиентов в яблочный сок: 1  
Название ингридиента: Яблоки  
Название 3 рецепта: яичница  
Сколько ингридиентов в яичница: 1  
Название ингридиента: Яйца  
Список рецептов, которыми можно порадовать гостей:  
 тосты  
 яблочный сок

#### 4.5.5 Контрольное задание. Использование множеств

**Задание.** Через сколько итераций функция `random.randint` выдаст повтор.

```
from random import randint

random_set = set()
while True:
    new_number = randint(1, 100)
    if new_number in random_set:
        break
    random_set.add(new_number)

print(len(random_set)+1)
```

15

#### 4.5.6 Задачи для самостоятельного выполнения

В программном решении вашей задачи требуется использовать конструкцию множество.

**Задача 1.** В озере водится несколько видов рыб. Три рыбака поймали рыб, представляющих некоторые из имеющихся видов. Определить: какие виды рыб есть у каждого рыбака; какие рыбы есть в озере, но нет ни у одного из рыбаков.

**Задача 2.** Задан некоторый набор товаров. Определить для каждого из товаров, какие из них имеются в каждом из  $n$  магазинов, какие товары есть хотя бы в одном магазине, и каких товаров нет ни в одном магазине.

**Задача 3.** Дан текст на русском языке. Напечатать в алфавитном порядке все гласные буквы, которые входят в каждое слово.

**Задача 4.** Дан текст на русском языке. Напечатать в алфавитном порядке все согласные буквы, которые входят только в одно слово.

**Задача 5.** Дан текст на русском языке. Напечатать в алфавитном порядке все согласные буквы, которые не входят ни в одно слово.

**Задача 6.** Дан текст из строчных латинских букв, за которыми следует точка. Напечатать все буквы, входящие в текст по одному разу.

**Задача 7.** Дан текст из строчных латинских букв, за которыми следует точка. Напечатать все буквы, входящие в текст не менее трех раз.

**Задача 8.** Дан текст на русском языке. Напечатать в алфавитном порядке все гласные буквы, которые входят в слова, состоящие из четного числа символов.

**Задача 9.** Дан текст на русском языке. Напечатать в алфавитном порядке все согласные буквы, которые входят хотя бы в одно слово.

**Задача 10.** Вывести в возрастающем порядке все цифры, не входящие в десятичную запись некоторого натурального числа  $n$ .

## § 4.6 Словари

### 4.6.1 Словарь - наиболее распространенный тип данных

В массиве или в списке **индекс** – это **целое число**. Традиционной является следующая ситуация:

```
>>> Days = [ 'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday' ]
>>> Days[0]
'Sunday'
>>> Days[1]
'Monday'
```

А как реализовать обратное соответствие?

```
>>> Days[ 'Sunday' ]
0
>>> Days[ 'Monday' ]
1
```

При помощи списка или массива это сделать невозможно, нужно использовать **ассоциативный массив** или **словарь**.

**В словаре индекс может быть любого неизменяемого типа!** Индексы, как и сами хранимые значения, задаются явно:

```
Days = {
    'Sunday': 0,
    'Monday': 1,
    'Tuesday': 2,
    'Wednesday': 3,
    'Thursday': 4,
    'Friday': 5,
    'Saturday': 6
}
>>> Days[ 'Sunday' ]
0
>>> Days[ 'Monday' ]
1
```

**Словарь (dict)** – это **неупорядоченная** коллекция пар элементов «ключ-значение». В разных языках синонимом отображений являются термины *отображение*, *хеш-таблица* или *ассоциативный массив*. Словари в языке Python – наиболее распространенный тип данных. Они буквально везде (в классах, модулях, функциях), поэтому сделаны максимально эффективными.

Словари в Python представлены единственным типом `dict` (словарь), в котором **в качестве ключа** может выступать произвольный **неизменяемый тип данных** (любое хешируемое значение: целые и действительные числа, строки, кортежи), **значением элемента** словаря может быть **любой тип данных** (число, строка, список, кортеж, словарь и пр.), в том числе и изменяемый. Ключом в словаре не может быть множество, но может быть элемент типа `frozenset`.

**Хешируемые объекты** – объекты, которые имеют метод `__hash__()` и могут участвовать в операциях сравнения на равенство с помощью метода `__eq__()`. Метод `__hash__()` возвращает одно и то же значение объекта на протяжении его жизненного цикла.

Т.к. словари являются неупорядоченными коллекциями, то к ним **не применяется понятие индекса элемента и не применяется операция извлечения среза**.

Значения ключей уникальны, двух одинаковых ключей в словаре быть не может. А вот значения могут быть одинаковыми. Если в словарь добавляется пара «ключ-значение» с ключом, который уже присутствует в словаре, в результате происходит замена значения существующего ключа новым значением.

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. Словари позволяют получить значение по ключу за константное время, очень быстро. Это достигается с помощью алгоритма хеширования.

## 4.6.2 Пустой словарь

**Пустой словарь** создается с помощью `{}` (вот почему фигурные скобки нельзя использовать для создания пустого множества) или функции `dict()`:

```
>>> {}
{}
>>> dict()
{}
```

В [17]:

```
d = {}
print(d)
```

```
{}
```

## 4.6.3 Создание словаря

Создать словарь можно несколькими способами.

### 4.6.3.1 С помощью {}

**Способ 1.** С помощью **фигурных скобок** с перечислением элементов в виде `ключ: значение`. Если на момент написания программы известны элементы словаря, то последний создается так:

```
<имя словаря> = {<ключ 1>: <значение 1>, <ключ 2>: <значение 2>, ...}
```

**Пример:**

```
capitals = {'Russia': 'Moscow', 'Ukraine': 'Kiev', 'USA': 'Washington'}
browser = {'g': 'Google', 'm': 'Mail', 'y': 'Yandex', 'o': 'Opera'}
```

В [5]:

```
capitals = {'Russia': 'Moscow', 'Ukraine': 'Kiev', 'USA': 'Washington'}  
browser = {'g': 'Google', 'm': 'Mail', 'y': 'Yandex', 'o': 'Opera'}  
print(capitals)  
print(browser)
```

```
{'Russia': 'Moscow', 'Ukraine': 'Kiev', 'USA': 'Washington'}  
{'g': 'Google', 'm': 'Mail', 'y': 'Yandex', 'o': 'Opera'}
```

#### 4.6.3.2 С помощью конструктора dict()

**Способ 2.** Если тип всех ключей – строковый и они не содержат пробелов (ключи являются допустимыми идентификаторами), то для создания словаря удобно использовать функцию `dict()`, передав набор пар `ключ=значение` (где ключ – строка без пробелов). В этом случае ключи можно указывать без кавычек.

**Пример:**

```
capitals = dict(Russia = 'Moscow', Ukraine = 'Kiev', USA = 'Washington')  
browser = dict(g="Google", m='Mail', y='Yandex', o='Opera')
```

В [7]:

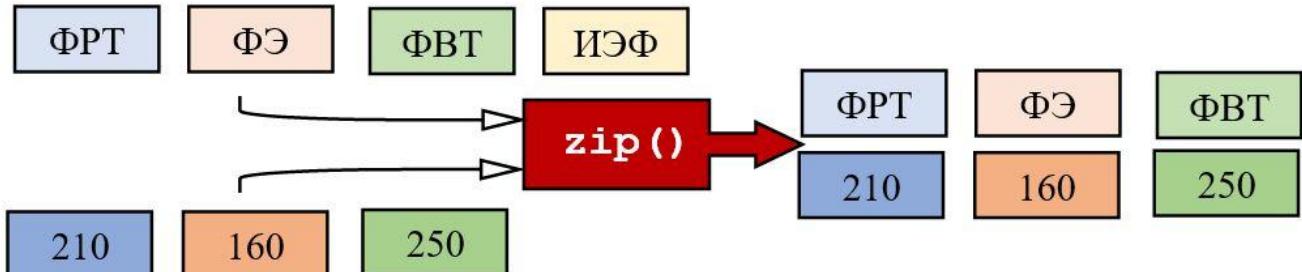
```
capitals = dict(Russia = 'Moscow', Ukraine = 'Kiev', USA = 'Washington')  
browser = dict(g="Google", m='Mail', y='Yandex', o='Opera')  
print(capitals)  
print(browser)
```

```
{'Russia': 'Moscow', 'Ukraine': 'Kiev', 'USA': 'Washington'}  
{'g': 'Google', 'm': 'Mail', 'y': 'Yandex', 'o': 'Opera'}
```

#### 4.6.3.3 С помощью функции zip()

**Способ 3.** С помощью функции `zip()` (от англ. `zip` – застежка-молния). Команда `zip` создает последовательность кортежей объектов из нескольких списков.

Функция `zip(A, B)` работает с итерируемыми объектами A и B **не обязательно одинаковой длины** и возвращает итерируемый объект пар кортежей из элементов A и B. Т.е. если ей передать два списка, то на первой итерации будет возвращен кортеж из первых элементов обоих списков, на второй кортеж из вторых элементов списков и т.д. (функция `zip()` делает все на лету, не занимает место в памяти). Функция берет два списка, а отдает список кортежей. Функция `zip` возвращает итератор, который останавливается, когда исчерпывается самая короткая последовательность.



Применив конструктор словаря к итерируемому списку кортежей, получим словарь:

```
D = dict(zip(Key, Val))
```

где Key – список ключей; Val – список значений.

В [8]:

```
key = ['Russia', 'Ukraine', 'USA']
val = ['Moscow', 'Kiev', 'Washington']
capitals = dict(zip(key, val))
print(capitals)
```

```
{'Russia': 'Moscow', 'Ukraine': 'Kiev', 'USA': 'Washington'}
```

В [10]:

```
browser = dict(zip(['g', 'm', 'y', 'o'], ["Google", 'Mail', 'Yandex', 'Opera']))
print(browser)
```

```
{'g': 'Google', 'm': 'Mail', 'y': 'Yandex', 'o': 'Opera'}
```

Для создания словаря с некоторым набором начальных значений можно также использовать следующие конструкции:

```
capitals = dict([("Russia", "Moscow"), ("Ukraine", "Kiev"), ("USA", "Washington")]
)
capitals = dict({"Russia": "Moscow", "Ukraine": "Kiev", "USA": "Washington"})
```

В [13]:

```
capitals = dict({"Russia": "Moscow", "Ukraine": "Kiev", "USA": "Washington"})
print(capitals)
```

```
{'Russia': 'Moscow', 'Ukraine': 'Kiev', 'USA': 'Washington'}
```

#### 4.6.3.4 С помощью метода fromkeys()

**Способ 4.** С использованием метода **fromkeys()**. Метод `d.fromkeys(s[, val])` возвращает словарь типа `dict`, ключами которого являются элементы последовательности `s`, а значениями либо `None`, либо `val`, если аргумент `val` определен. Этот способ удобно использовать для инициализации пустых словарей или словарей со значением элементов, которые по умолчанию должны быть равны какому-то значению.

```
>>> fields = ('name', 'age', 'job', 'pay')
>>> record = dict.fromkeys(fields, '?')
>>> record
{'name': '?', 'age': '?', 'job': '?', 'pay': '?'}
```

В [1]:

```
fields = ('name', 'age', 'job', 'pay')
record = dict.fromkeys(fields, '?')
print(record)
```

```
{'name': '?', 'age': '?', 'job': '?', 'pay': '?'}
```

#### 4.6.3.5 С помощью генератора словарей

Способ 5. Также можно использовать **генератор словарей**:

```
>>> d = {a: a ** 2 for a in range(7)}
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
```

В [19]:

```
d = {a: a ** 2 for a in range(7)}
print(d)
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
```

В [41]:

```
rec = [("Russia", "Moscow"), ("Ukraine", "Kiev"), ("USA", "Washington")]
capitals = {k: v for k, v in rec}
print(capitals)
```

```
{'Russia': 'Moscow', 'Ukraine': 'Kiev', 'USA': 'Washington'}
```

В [60]:

```
# Заменить в словаре все значения на их квадраты
dct = {'one': 10, 'two': -5, 'three': 12, 'four': -9}

dct = {key: value**2 for key, value in dct.items()}
print(dct)
```

```
{'one': 100, 'two': 25, 'three': 144, 'four': 81}
```

Несмотря на то что **словарь – неупорядоченный набор данных** (порядок его элементов может быть произвольным) порядок вставленных элементов по ключу в словаре в новых версиях языка (старше 3.6) сохраняется:

В [49]:

```
d = {}
d['tree'] = 3
d['two'] = 2
d['one'] = 1
print(d)
▼ for k, v in d.items(): # аналог enumerate() для словаря
    print(k, v)
```

```
{'tree': 3, 'two': 2, 'one': 1}
tree 3
two 2
one 1
```

#### 4.6.4 Ручной ввод ключей и значений словаря

Для ввода ключей и элементов словаря можно использовать инструкцию цикла:

```
d = {}
n = int(input('Введите количество элементов словаря '))
for i in range(n):
    key = input('Введите ключ очередного элемента словаря ')
    val = input('Введите значение очередного элемента словаря ')
    d[key] = val
```

В [1]:

```
d = {}
n = int(input('Введите количество элементов словаря '))
▼ for i in range(n):
    key = input('Введите ключ очередного элемента словаря ')
    val = input('Введите значение очередного элемента словаря ')
    d[key] = val
print(d)
```

```
Введите количество элементов словаря 2
Введите ключ очередного элемента словаря Россия
Введите значение очередного элемента словаря Москва
Введите ключ очередного элемента словаря Англия
Введите значение очередного элемента словаря Лондон
{'Россия': 'Москва', 'Англия': 'Лондон'}
```

#### 4.6.5 Доступ к элементам словаря и отображение его содержимого

Словари, как и все коллекции, поддерживают протокол итерации, поэтому мы **можем итерироваться по словарю**, например, с помощью метода `for`. Итератор словаря на каждой итерации в цикле `for` воспроизводит по одному ключу. Поэтому для перебора элементов словаря по ключу можно использовать следующий синтаксис:

```
capitals = dict(Russia = 'Moscow', Ukraine = 'Kiev', USA = 'Washington')
for key in capitals:
    print(key, '=>', capitals[key])
```

В [24]:

```
capitals = dict(Russia = 'Moscow', Ukraine = 'Kiev', USA = 'Washington')
for key in capitals:
    print(key, '=>', capitals[key])
```

```
Russia => Moscow
Ukraine => Kiev
USA => Washington
```

Обратите внимание, по умолчанию происходит **перебор (итерирование) по ключам словаря**. Также существует симметричный метод `keys`, который возвращает оператор ключей. Если нужно итерироваться не по ключам, как по умолчанию, а по значениям, используется метод `values`, который возвращает именно значения. Если нам нужно итерироваться сразу по ключам и значениям можно использовать метод словаря `items`, который возвращает ключи и значения.

Таким образом, **при работе со словарями можно использовать методы:**

- `keys` – возвращает представление ключей всех элементов;
- `values` – возвращает представление всех значений;
- `items` – возвращает представление всех пар (кортежей) из ключей и значений.

**Пример.** Использование метода `keys()`. Перебор всех элементов по ключу.

```
for key in capitals.keys():
    print(key, '=>', Capitals[key])
```

**Пример.** Использование метода `values()`. Перебор значений элементов словаря.

```
for val in capitals.values():
    print(val)
```

**Пример.** Использование метода `items()`. Перебор значений ключей и значений всех элементов словаря.

```
for key, val in capitals.items():
    print('{} => {}'.format(key, val))
```

**Пример.** Эквивалентный способ обхода пар «ключ-значение»:

```
for item in capitals.items():
    print('{} => {}'.format(item[0], item[1]))
```

В [27]:

```
# Использование метода items(). Перебор значений ключей и значений всех элементов словаря
capitals = dict(Russia = 'Moscow', Ukraine = 'Kiev', USA = 'Washington')
for key, val in capitals.items():
    print('{} => {}'.format(key, val))
```

```
Russia => Moscow
Ukraine => Kiev
USA => Washington
```

Методы `dict.items()`, `dict.keys()` и `dict.values()` возвращают **представления словарей**.

**Представление словаря** – это в действительности итерируемый объект, доступный только для чтения и хранящий элементы, ключи или значения словаря в зависимости от того, какое представление было запрошено. Однако между представлениями и обычными итерируемыми объектами есть два различия. Одно из них заключается в том, что если словарь, для которого было получено представление, изменяется, то представление будет отражать эти изменения.

В [28]:

```
# В примере меняем значение в словаре (по ключу), при этом поменяется и представление
capitals = {'Russia': 'Moscow', 'Ukraine': 'Kiev', 'USA': 'Washington'}
key = capitals.keys()
val = capitals.values()
print(key)
print(val)

capitals['Russia'] = 'Sankt Petersburg'
print(key)
print(val)

dict_keys(['Russia', 'Ukraine', 'USA'])
dict_values(['Moscow', 'Kiev', 'Washington'])
dict_keys(['Russia', 'Ukraine', 'USA'])
dict_values(['Sankt Petersburg', 'Kiev', 'Washington'])
```

Другое отличие состоит в том, что **представления ключей и элементов поддерживают** некоторые **операции, свойственные множествам** (пересечение &, объединение +, разность -, симметрическая разность ^).

В [31]:

```
# Пример использования операций со множествами для ключей словаря
shengen = {
    'Avstriya': 'Вена',
    'Daniya': 'Копенгаген',
    'Franciya': 'Париж',
    'Ispaniya': 'Мадрид',
    'Niderlandy': 'Амстердам',
    'Portugaliya': 'Лиссабон',
    'Italiya': 'Рим',
    'Latviya': 'Рига',
    'Mal\'ta': 'Валлетта',
    'CHekhiya': 'Прага',
    'Sloveniya': 'Любляна',
    'Norvegiya': 'Осло',
    'SHvejcariya': 'Берн'
}
ES={

    'Avstriya': 1995, 'Bel\'giya': 1957, 'Bulgariya': 2007,
    'Velikobritaniya': 1973, 'Vengriya': 2004, 'Germaniya': 1957,
    'Greciya': 1981, 'Daniya': 1973, 'Irlandiya': 1973, 'Ispaniya': 1986,
    'Italiya': 1957, 'Kipr': 2004, 'Latviya': 2004, 'Litva': 2004,
    'Lyuksemburg': 1957, 'Mal\'ta': 2004, 'Niderlandy': 1957,
    'Pol\'sha': 2004, 'Portugaliya': 1986, 'Rumyniya': 2007,
    'Slovakiya': 2004, 'Sloveniya': 2004, 'Finlyandiya': 1995,
    'Franciya': 1957, 'Horvatiya': 2013, 'CHekhiya': 2004,
    'SHveciya': 1995, 'Estoniya': 2004
}

print('Столицы стран, входящих в шенген и ЕС:')
matches = shengen.keys() & ES.keys()
for stolitsa in matches:
    print(' ', shengen[stolitsa])

print('Столицы стран, входящих в шенген, но не входящих в ЕС:')
nomatches = shengen.keys() - ES.keys()
for stolitsa in nomatches:
    print(' ', shengen[stolitsa])
```

Столицы стран, входящих в шенген и ЕС:

Лиссабон  
Париж  
Прага  
Брюсель  
Будапешт  
Вильнюс  
Любляна  
Мадрид  
Афины  
Копенгаген  
Валлетта  
Хельсинки  
Вена  
Люксембург  
Таллин  
Стокгольм  
Амстердам  
Рим  
Варшава

Берлин  
Братислава  
Рига

Столицы стран, входящих в шенген, но не входящих в ЕС:

Рейкьявик  
Берн  
Осло

**Замечание.** Учтите, что итерироваться по представлениям, изменяя сам словарь нельзя:

```
capitals = {'Russia': 'Moscow', 'Ukraine': 'Kiev', 'USA': 'Washington'}
for key in capitals.keys():
    if key == 'Ukraine':
        del capitals[key]      # <= ОШИБКА!!!!
print(capitals)
```

B [38]:

```
capitals = {'Russia': 'Moscow', 'Ukraine': 'Kiev', 'USA': 'Washington'}
for key in capitals.keys():
    if key == 'Ukraine':
        del capitals[key]
print(capitals)
```

```
RuntimeError                                     Traceback (most recent call last)
<ipython-input-38-9e1ca5b57c65> in <module>
      1 capitals = {'Russia': 'Moscow', 'Ukraine': 'Kiev', 'USA': 'Washington'}
      2 for key in capitals.keys():
----> 3     if key == 'Ukraine':
      4         del capitals[key]
      5 print(capitals)

RuntimeError: dictionary changed size during iteration
```

Подобного рода задачу **можно решить**, если в начале **скопировать представление в список (или множество)**:

```
capitals = {'Russia': 'Moscow', 'Ukraine': 'Kiev', 'USA': 'Washington'}
for key in list(capitals.keys()):
    if key == 'Ukraine':
        del capitals[key]
print(capitals)
```

```

capitals = {'Russia': 'Moscow', 'Ukraine': 'Kiev', 'USA': 'Washington'}
for key in list(capitals.keys()):
    if key == 'Ukraine':
        del capitals[key]
print(capitals)

```

```
{'Russia': 'Moscow', 'USA': 'Washington'}
```

## 4.6.6 Методы для работы со словарями

Рассмотрим операции, доступные для словарей.

*Таблица.* Операции, доступные для словарей ( dct – словарь, на котором вызывается метод)

Синтаксис метода	Описание метода	Трудоемкость
dct[key]	Получение элемента по ключу key . Если элемента с заданным ключом в словаре нет, то возникает исключение KeyError	O(1)
dct.get(key[, default])	Возвращает значение словаря для ключа key . Если ключ не существует, возвращается значение default или None	O(1)
dct[key] = value	Устанавливает значение словаря по ключу key . Если ключ не существует, он создается со значением value (добавляется новый элемент)	O(1)
key in dct	Проверить принадлежность ключа словарю dct	O(1)
key not in dct	То же, что not key in dct	O(1)
del dct[key]	Удаляет пару «ключ-значение» на основании ключа key . Возбуждает исключение KeyError , если такого ключа нет	O(1)
dct.pop(key)	Возвращает значение ключа key и удаляет из словаря элемент с ключом key или возбуждает исключение KeyError , если ключ key отсутствует в словаре dct	
dct.pop(key, default)	Возвращает значение ключа key и удаляет из словаря элемент с ключом key или возвращает значение default , если ключ key отсутствует в словаре dct	
dct.popitem()	Возвращает и удаляет пару (ключ, значение) с конца словаря dct или возбуждает исключение KeyError , если словарь dct пуст	
dct.items()	Возвращает представление всех пар «ключ-значение» в словаре dct	
dct.keys()	Возвращает представление всех ключей словаря dct	
dct.values()	Возвращает представление всех значений в словаре dct	
dct.clear()	Удаляет из словаря dct все элементы	
len(dct)	Возвращает количество пар ключ-значение, хранящихся в словаре dct	
dct.fromkeys(seq[, val])	Возвращает словарь типа dict , ключами которого являются элементы последовательности seq , а значениями либо None , либо val , если аргумент val определен	
dct.update(other)	Добавляет в словарь dct пары (ключ, значение) из other , которые отсутствуют в словаре dct , а для каждого ключа, который уже присутствует в словаре dct , выполняется замена соответствующим значением из other ; other может быть словарем, итерируемым объектом с парами (ключ, значение) или именованными аргументами	
dct.setdefault(key[, val])	То же, что и dict.get() за исключением того, что, если ключ key в словаре отсутствует, в словарь вставляется новый элемент с ключом key и со значением None или val , если аргумент val задан	
dct.copy()	Возвращает копию словаря dct	

**Замечание.** Присвоение по новому ключу расширяет словарь, присвоение по существующему ключу перезаписывает его, а попытка извлечения несуществующего ключа порождает исключение. Для избежания исключения можно действовать в соответствии с одной из двух стратегий:

- для получения элемента использовать метод `dict.get(key[, default])`.
- перехватывать и обрабатывать исключение.

В [46]:

```
# Создать словарь
dct = dict(key1='val1', key2='val2')

# Получить значение по ключу (вызывает исключение
# при отсутствии такого ключа в словаре)
print(dct['key1'])          # => val1

# Получить значение по ключу (НЕ вызывает исключение
# при отсутствии такого ключа в словаре)
print(dct.get('key1'))      # => val1
print(dct.get('key3'))      # => None

# Получить пары ключ-значение
print(dct.items())         # => dict_items([('key1', 'val1'), ('key2', 'val2')])

# Получить все ключи в словаре
print(dct.keys())          # => dict_keys(['key1', 'key2'])

# Получить значение по ключу, после чего удалить
print(dct.pop('key1'))      # => val1
print(dct)                  # => {'key2': 'val2'}

# Добавить новое значение в словарь
dct['key3'] = 'val3'
print(dct)                  # => {'key2': 'val2', 'key3': 'val3'}

# Получить последний элемент, после чего удалить
print(dct.popitem())        # => ('key3', 'val3')
print(dct)                  # => {'key2': 'val2'}
```

```
val1
val1
None
dict_items([('key1', 'val1'), ('key2', 'val2')])
dict_keys(['key1', 'key2'])
val1
{'key2': 'val2'}
{'key2': 'val2', 'key3': 'val3'}
('key3', 'val3')
{'key2': 'val2'}
```

**Пример использования `setdefault`.** Часто бывает необходимо не только попробовать проверить, существует ли ключ в словаре, но и в случае неудачи добавить эту новую пару ключ-значение. Для этого есть метод `setdefault`. Объявляем `unknown_dict`, который абсолютно пуст, и используя метод `setdefault` пытаемся взять какой-то ключ `key` и в случае неудачи добавим туда `default`. Происходит именно это, у нас в нашем пустом словаре получилось отображение `key`, `default` и

`default` вернулся к нам в результате вызова метода `setdefault`. Если мы попытаемся вызвать `setdefault` и в качестве дефолтного значения передаём `new_default`, у нас вернётся значение, которое уже лежит в словаре, значение `default`.

В [50]:

```
unknown_dict = {}
print(unknown_dict.setdefault('key', 'default'))      # => default
print(unknown_dict)                                    # => {'key': 'default'}
print(unknown_dict.setdefault('key', 'new_default'))  # => default
```

```
default
{'key': 'default'}
default
```

### Трудоемкость стандартных операций по работе со словарем

**Словарь отличается от списка** возможностью доступа к элементам по ключу, а не позиции. На данный момент наиболее важной характеристикой является то, что получение и присваивание элемента в словаре являются операциями за  $O(1)$ . Словари были созданы специально для того, чтобы как можно быстрее получить и установить значения по ключу.

Другая важная операция словаря – проверка наличия ключа в словаре. Операция `contains` (реализует оператор `in`) также работает за  $O(1)$  (в случае со списками это занимало  $O(N)$ ), потому что проверка для данного ключа подразумевает простое получение элемента по ключу, которое делается за  $O(1)$ .

Для быстрого получения адреса памяти ключа используется функция хэширования от его содержимого. Ключами в словарях могут быть следующие типы данных: строки (строки неизменяемы и часто используются в качестве ключей), целые числа, `None`, кортежи.

## 4.6.7 Когда использовать словари

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

## 4.6.8 Вложенные словари

Элементами словаря также могут быть список, кортеж, словарь.

### Пример вложенного словаря

```
worker = {'name': {'first': 'Александр', 'last': 'Иванов'},  
          'age': 42,  
          'job': ['редактор', 'корректор'],  
          'pay': (50000, 30000) }
```

Эта запись содержит вложенные структуры, поэтому для доступа к более низкому уровню используют двойные индексы:

```
print('Фамилия:', worker['name']['last'])  
print('    Имя:', worker['name']['first'])  
print('Оклад по основной должности: ', worker['pay'][0])  
print('Занимаемые должности:'.center(30,"*"))  
for job in worker['job']:  
    print("{0:>20}".format(job))
```

В [2]:

```
▼ worker = {'name': {'first': 'Александр', 'last': 'Иванов'},  
          'age': 42,  
          'job': ['редактор', 'корректор'],  
          'pay': (50000, 30000) }  
print('Фамилия:', worker['name']['last'])  
print('    Имя:', worker['name']['first'])  
print('Оклад по основной должности: ', worker['pay'][0])  
print('Занимаемые должности:'.center(30,"*"))  
▼ for job in worker['job']:  
    print("{0:>20}".format(job))
```

```
Фамилия: Иванов  
    Имя: Александр  
Оклад по основной должности: 50000  
****Занимаемые должности:****  
    редактор  
    корректор
```

#### 4.6.9 Модуль PrettyTable в Python, вывод табличных данных

**Модуль prettytable** – полезен при создании простых таблиц и вывода их в терминал или текстовый файл.

Возможности модуля prettytable :

- установка ширины заполнения столбца, выравнивание текста или граница таблицы;
- сортировка данных;
- выбор отображения столбцов и строк в окончательном выводе;
- чтение данных из CSV, HTML или базы данных;
- вывод данных в ASCII или HTML.

**Установка модуля PrettyTable** Для того, чтобы установить модуль PrettyTable необходимо выполнить команду:

```
pip install -U prettytable
```

В [1]:

```
pip install -U prettytable
```

```
Collecting prettytable
  Downloading prettytable-2.5.0-py3-none-any.whl (24 kB)
Requirement already satisfied: wcwidth in c:\programdata\anaconda3\lib\site-packages (from prettytable) (0.2.5)
Installing collected packages: prettytable
Successfully installed prettytable-2.5.0
Note: you may need to restart the kernel to use updated packages.
```

#### 4.6.9.1 Создание таблицы и добавление данных

**Шаг 1.** Создание таблицы. Сначала, необходимо создать экземпляр `PrettyTable()`:

```
# импорт установленного модуля
from prettytable import PrettyTable
# создание экземпляра
mytable = PrettyTable()
```

**Шаг 2.** Задание названия полей таблицы. Для того, чтобы установить имена полей, используется атрибут `PrettyTable.field_names`:

```
mytable.field_names = ['Colum_1', 'Colum_2', 'Colum_3', 'Colum_N']
```

**Шаг 3.** Добавление данных. Затем нужно добавлять в созданный экземпляр некоторые данные. Есть несколько вариантов добавления данных.

- *добавление данных построчно*: можно добавлять данные по одной строке за раз, используя метод `PrettyTable.add_row()`;
- *добавление сразу всех строк*: когда есть список строк, то можно добавить их за один раз с помощью метода `PrettyTable.add_rows()`.

**Шаг 4.** Вывод таблицы в терминал. Для вывода таблицы в терминал воспользоваться методом `print(mytable)`, где `mytable` – имя экземпляра.

```
print(mytable)
```

**Пример.** Рассмотрим задачу табулирования функции  $y = x^2$  на отрезке  $[-2; 1]$   $h = 0.3$ . Представим результаты табуляции в виде таблицы (используя построчный вывод).

```

from prettytable import PrettyTable

mytable = PrettyTable(title='Табулирование функции', min_table_width=30)
mytable.field_names = ['x', 'f(x)']

x0 = -2
xn = 1
h = 0.3

x = x0
while x <= xn + h/6:
    y = x * x
    mytable.add_row([round(x, 2), round(y, 2)])
    x = x + h

print(mytable)

```

Табулирование функции	
x	f(x)
-2	4
-1.7	2.89
-1.4	1.96
-1.1	1.21
-0.8	0.64
-0.5	0.25
-0.2	0.04
0.1	0.01
0.4	0.16
0.7	0.49
1.0	1.0

**Пример.** Рассмотрим задачу табулирования функции  $y = x^2$  на отрезке  $[-2; 1]$   $h = 0.3$ . Представим результаты табуляции в виде таблицы (используя вывод всех строк за раз).

```

from prettytable import PrettyTable

mytable = PrettyTable()

mytable.field_names = ['x', 'f(x)']

x0 = -2
xn = 1
h = 0.3
x = x0
data = []
while x <= xn + h/6:
    data.append([round(x, 2), round(x * x, 2)])
    x = x + h

mytable.add_rows(data)
print(mytable)

```

x	f(x)
-2	4
-1.7	2.89
-1.4	1.96
-1.1	1.21
-0.8	0.64
-0.5	0.25
-0.2	0.04
0.1	0.01
0.4	0.16
0.7	0.49
1.0	1.0

#### 4.6.9.2 Создание пользовательского стиля отображения таблицы.

Модуль `prettytable` имеет несколько параметров стиля, которые управляют различными аспектами отображения таблиц. У пользователя есть свобода установить каждый из этих параметров индивидуально в соответствии с предпочтениями. Метод `PrettyTable.set_style()` просто делает это автоматически, используя встроенные стили: `MSWORD_FRIENDLY` , `PLAIN_COLUMNS` , `DEFAULT` , которые необходимо предварительно импортировать:

```

from prettytable import PrettyTable
from prettytable import MSWORD_FRIENDLY
from prettytable import PLAIN_COLUMNS
from prettytable import RANDOM
from prettytable import DEFAULT

table = PrettyTable(['Номер', 'Имя облака', 'IP-адрес'])
table.add_row(['1', 'server01', '172.16.0.1'])
table.add_row(['2', 'server03', '172.16.0.3'])
table.add_row(['3', 'server02', '172.16.0.2'])
table.add_row(['4', 'server09', '172.16.0.9'])

table.set_style(PLAIN_COLUMNS) # MSWORD_FRIENDLY, PLAIN_COLUMNS, DEFAULT

print(table)

```

В [64]:

```

from prettytable import PrettyTable
from prettytable import MSWORD_FRIENDLY
from prettytable import PLAIN_COLUMNS
from prettytable import RANDOM
from prettytable import DEFAULT

table = PrettyTable(['Номер', 'Имя облака', 'IP-адрес'])
table.add_row(['1', 'server01', '172.16.0.1'])
table.add_row(['2', 'server03', '172.16.0.3'])
table.add_row(['3', 'server02', '172.16.0.2'])
table.add_row(['4', 'server09', '172.16.0.9'])

table.set_style(PLAIN_COLUMNS) # MSWORD_FRIENDLY, PLAIN_COLUMNS, DEFAULT

print(table)

```

Номер	Имя облака	IP-адрес
1	server01	172.16.0.1
2	server03	172.16.0.3
3	server02	172.16.0.2
4	server09	172.16.0.9

Если необходимо напечатать таблицу с другим стилем несколько раз, то можно установить свой вариант на длительный срок, просто изменив соответствующие атрибуты экземпляра класса `PrettyTable`:

Таблица. Параметры для управления стилями `PrettyTable`

Параметр	Описание, тип параметра, пример
encoding	Схема кодирования Unicode, используемая для декодирования любого закодированного ввода
title	Необязательный заголовок таблицы
field_names	Список или кортеж имен полей
fields	Список или кортеж имен полей для включения в дисплеи
start	Индекс первой строки данных для включения в вывод

Параметр	Описание, тип параметра, пример
end	Индекс последней строки данных для включения в вывод PLUS ONE (стиль среза списка)
header	Распечатать заголовок, показывающий имена полей ( True или False )
header_style	Стилизация, применяемая к именам полей в заголовке («шапка», «заголовок», «верхний», «нижний» или «Нет»)
border	Распечатать рамку вокруг столба ( True или False )
hrules	Управляет печатью горизонтальных линий после строк, допустимые значения: FRAME , HEADER , ALL , NONE
vrules	Управляет печатью вертикальных линий между столбцами, допустимые значения: FRAME , ALL , NONE
int_format	Управляет форматированием целочисленных данных, работает так: print('% int_format d' % data)
float_format	Управляет форматированием данных с плавающей запятой, работает так: print('% float_format f' % data)
min_table_width	Минимальная желаемая ширина таблицы, в символах
max_table_width	Максимальная желаемая ширина таблицы, в символах
min_width	Минимальная желаемая ширина поля, в символах
max_width	Максимальная желаемая ширина поля, в символах
padding_width	Количество пробелов по обе стороны от данных столбца (используется только в том случае, если левое и правое отступы равны None )
left_padding_width	Количество пробелов слева от данных столбца
right_padding_width	Количество пробелов справа от данных столбца
vertical_char	Односимвольная строка, используемая для рисования вертикальных линий (по умолчанию `)
horizontal_char	Односимвольная строка, используемая для рисования горизонтальных линий (по умолчанию – )
junction_char	Односимвольная строка, используемая для рисования стыков линий (по умолчанию + )
top_junction_char	Односимвольная строка, используемая для рисования стыков верхней линии
bottom_junction_char	Односимвольная строка, используемая для рисования стыков нижней линии
right_junction_char	Односимвольная строка, используемая для рисования стыков правой линии
left_junction_char	Односимвольная строка, используемая для рисования левых стыков линий
top_right_junction_char	Односимвольная строка, используемая для рисования стыков верхних и правых линий
top_left_junction_char	Односимвольная строка, используемая для рисования стыков верхних левых линий
bottom_right_junction_char	Односимвольная строка, используемая для рисования нижних и правых стыков линий
bottom_left_junction_char	Односимвольная строка, используемая для рисования нижних левых стыков линий
sortby	Имя поля для сортировки строк
align	Выравнивание по умолчанию для каждого столбца: None , l , c или r
valign	Значение по умолчанию для каждой строки: None , t , m или b

Для построения таблицы могут пригодиться **графические символы кодовой таблицы ASCII с кодами:**

- ⠉ 218
- | 179
- ⠄ 195

- ⌈ 191
- ⌉ 194
- ⌊ 180
- ⌋ 196
- ⌂ 192
- ⌃ 197
- ⌄ 217
- ⌅ 193

Для получения графического символа используется комбинация клавиш ALT + код .

**Обратите внимание:** если вы знаете, какие параметры стиля нужны в момент создания таблицы, то можно указать их, используя ключевые аргументы для конструктора. Например, следующие два блока кода эквивалентны:

```
mytable = PrettyTable()  
mytable.border = False  
mytable.header = False  
mytable.padding_width = 5  
  
# эквивалентно  
mytable = PrettyTable(border=False, header=False, padding_width=5)
```

В [65]:



```
from prettytable import PrettyTable

mytable = PrettyTable(title='Информация по европейским странам',
                      border=True, header = True, vertical_char="|",
                      horizontal_char="-", junction_char="+",
                      left_junction_char='+', right_junction_char='|',
                      top_left_junction_char='Г', top_right_junction_char='Л',
                      top_junction_char='Т', bottom_junction_char='Д',
                      bottom_right_junction_char='Ј',
                      bottom_left_junction_char='Л',
                      start=0, #end=3
                     )

country = {
    'Avstriya': ['Австрия', 'Вена', 8.46, 83_879],
    'Daniya': ['Дания', 'Копенгаген', 5.82, 43_094],
    'Finlyandiya': ['Финляндия', 'Хельсинки', 5.54, 338_145],
    'Franciya': ['Франция', 'Париж', 68.08, 632_734],
    'Greciya': ['Греция', 'Афины', 10.74, 131_957],
    'Ispaniya': ['Испания', 'Мадрид', 46.6, 505_990],
    'Germaniya': ['Германия', 'Берлин', 83.02, 357_385],
    'Italiya': ['Италия', 'Рим', 60.32, 302_073],
    'Chekhiya': ['Чехия', 'Прага', 10.7, 78_866],
    'Vengriya': ['Венгрия', 'Будапешт', 9.73, 93_036],
    'SHvejcariya': ['Швейцария', 'Берн', 8.56, 41_284]
}

mytable.field_names = ['Страна', 'Столица', 'Численность', 'Площадь']
mytable.align['Страна'] = 'l'
mytable.align['Столица'] = 'c'
mytable.align['Численность'] = 'r'
mytable.align['Площадь'] = 'r'
# mytable.sortby = 'Страна'

for data in country.values():
    mytable.add_row(data)

print(mytable)
```

Информация по европейским странам			
Страна	Столица	Численность	Площадь
Австрия	Вена	8.46	83879
Дания	Копенгаген	5.82	43094
Финляндия	Хельсинки	5.54	338145
Франция	Париж	68.08	632734
Греция	Афины	10.74	131957
Испания	Мадрид	46.6	505990
Германия	Берлин	83.02	357385
Италия	Рим	60.32	302073
Чехия	Прага	10.7	78866
Венгрия	Будапешт	9.73	93036
Швейцария	Берн	8.56	41284

#### 4.6.9.3 Добавление данных колонками

Также можно добавлять данные по одному столбцу за раз. Для этого необходимо использовать метод `PrettyTable.add_column()`, который позволяет добавить столбец в таблицу. Он принимает два обязательных аргумента – строку, которая является именем поля таблицы добавляемого столбца, и список или кортеж, содержащий данные столбца.

**Синтаксис:** `mytable.add_column(fieldname, column, align='c', valign='t')`

где:

- `fieldname` – имя поля, которое будет содержать новый столбец данных;
- `column` – столбец данных, должен быть список (кортеж) с таким количеством элементов, сколько строк в таблице;
- `align` – желаемое горизонтальное выравнивание для столбца – `l` для выравнивания слева, `c` для выравнивания по центру и `r` для выравнивания справа;
- `valign` – желаемое вертикальное выравнивание для столбца – `t` для выравнивания по верху, `m` для выравнивания по середине и `b` для выравнивания по низу.

**Пример.** Добавление данных в PrettyTable колонками.

```
from prettytable import PrettyTable
mytable = PrettyTable()

# Добавление колонки таблицы с именем 'Страна'
mytable.add_column('Страна', ['Франция', 'Германия', 'Италия', 'Испания', 'Чехия'],
], align='l')

# Добавление колонки таблицы с именем 'Столица'
mytable.add_column('Столица', ['Париж', 'Берлин', 'Рим', 'Мадрид', 'Прага'], align=
='c')

# Добавление колонки таблицы с именем 'Площадь'
mytable.add_column('Площадь', [632_734, 357_385, 302_073, 505_990, 78_866], align=
'r')

print(mytable)
```

```
from prettytable import PrettyTable
mytable = PrettyTable()

# Добавление колонки таблицы с именем 'Страна'
mytable.add_column('Страна', ['Франция', 'Германия', 'Италия', 'Испания', 'Чехия'], align='c')

# Добавление колонки таблицы с именем 'Столица'
mytable.add_column('Столица', ['Париж', 'Берлин', 'Рим', 'Мадрид', 'Прага'], align='c')

# Добавление колонки таблицы с именем 'Площадь'
mytable.add_column('Площадь', [632_734, 357_385, 302_073, 505_990, 78_866], align='r')

print(mytable)
```

Страна	Столица	Площадь
Франция	Париж	632734
Германия	Берлин	357385
Италия	Рим	302073
Испания	Мадрид	505990
Чехия	Прага	78866

#### 4.6.9.4 Импорт данных из файла CSV

Если данные таблицы хранятся в файле csv, то можно прочитать эти данные и добавить в таблицу PrettyTable() следующим образом:

```
# импорт загрузчика `from_csv`
from prettytable import from_csv

with open("myfile.csv") as fp:
    # создание таблицы из `myfile.csv`
    mytable = from_csv(fp)
print(mytable)
```

```
from prettytable import from_csv
with open('Data/PrettyTable.csv', encoding='UTF-8') as fp:
    mytable = from_csv(fp)
print(mytable)
```

Страна	Столица	Площадь
Франция	Париж	632734
Германия	Берлин	357385
Италия	Рим	302073
Испания	Мадрид	505990
Чехия	Прага	78866

#### 4.6.9.5 Пример использования модуля prettytable

**Пример.** Дополнить таблицу со странами информацией о плотности населения. Отсортировать выводимые данные по убыванию плотности.

В [19]:

```
from prettytable import PrettyTable

mytable = PrettyTable(title='Информация по европейским странам',
                      border=True, header = True, vertical_char="|",
                      horizontal_char="-", junction_char="+",
                      left_junction_char='+', right_junction_char='|',
                      top_left_junction_char='Г', top_right_junction_char='Л',
                      top_junction_char='Т', bottom_junction_char='Д',
                      bottom_right_junction_char='J',
                      bottom_left_junction_char='L', start=0
                     )

country = {
    'Avstriya': ['Австрия', 'Вена', 8.46, 83_879],
    'Daniya': ['Дания', 'Копенгаген', 5.82, 43_094],
    'Finlyandiya': ['Финляндия', 'Хельсинки', 5.54, 338_145],
    'Franiya': ['Франция', 'Париж', 68.08, 632_734],
    'Greciya': ['Греция', 'Афины', 10.74, 131_957],
    'Ispaniya': ['Испания', 'Мадрид', 46.60, 505_990],
    'Germaniya': ['Германия', 'Берлин', 83.02, 357_385],
    'Italiya': ['Италия', 'Рим', 60.32, 302_073],
    'Chekhiya': ['Чехия', 'Прага', 10.7, 78_866],
    'Vengriya': ['Венгрия', 'Будапешт', 9.73, 93_036],
    'SHvejcariya': ['Швейцария', 'Берн', 8.56, 41_284]
}

mytable.field_names = ['Страна', 'Столица', 'Численность', 'Площадь']
mytable.align['Страна'] = 'l'
mytable.align['Столица'] = 'c'
mytable.align['Численность'] = 'r'
mytable.align['Площадь'] = 'r'

population_density = []
for data in country.values():
    mytable.add_row(data)
    population_density.append(data[2] * 1_000_000 / data[3])

# Добавление колонки таблицы с именем "Плотность"
mytable.add_column('Плотность', population_density, align='r')
mytable.float_format["Плотность"] = ".2"
mytable.reversesort=True
mytable.sortby = 'Плотность'

print(mytable)
```

Информация по европейским странам

Страна	Столица	Численность	Площадь	Плотность
Германия	Берлин	83.02	357385	232.30
Швейцария	Берн	8.56	41284	207.34
Италия	Рим	60.32	302073	199.69
Чехия	Прага	10.7	78866	135.67
Дания	Копенгаген	5.82	43094	135.05
Франция	Париж	68.08	632734	107.60
Венгрия	Будапешт	9.73	93036	104.58
Австрия	Вена	8.46	83879	100.86
Испания	Мадрид	46.6	505990	92.10

Греция	Афины	10.74	131957	81.39	
Финляндия	Хельсинки	5.54	338145	16.38	

## 4.6.10 Примеры решения задач

**Пример.** Дано число  $N$  – количество бюллетеней. Далее следуют  $N$  строчек с фамилиями кандидатов, взятых из бюллетеней. Требуется найти и определить победителя.

**Комментарии.** Фактически требуется подсчитать частоту появления фамилии в списке. Заранее кандидаты не известны. Для сортировки воспользуемся функцией `sorted`, которую применим к списку `list(votes.items())`. Этот список будет представлять собой список кортежей (`tuple`-ов), например: `[('Письменный', 1), ('Краснов', 1), ('Берман', 3)]`. Если отсортировать `list(votes.items())`, то сортировка произойдёт по первому значению (фамилии кандидата). Сортировку по второму значению выполним с использованием модуля `operator` и функции `sorted`. Чтобы сортировать по второму значению в нашем `tuple`-е, мы можем в качестве аргумента `key` передать туда `operator.itemgetter` и написать единичку, потому что нас интересует именно первый индекс. Так как нам нужны самые популярные слова, мы используем `reverse=True`.

В [58]:

```
from operator import itemgetter

votes = {}
n = int(input("Введите число бюллетеней "))
for i in range(n):
    name = input("Фамилия кандидата: ")
    votes[name] = votes.pop(name, 0) + 1

bests = sorted(list(votes.items()), key=itemgetter(1), reverse=True)
for best in bests:
    print('{} => {}'.format(best[0], best[1]))
```

```
Введите число бюллетеней 5
Фамилия кандидата: Иванов
Фамилия кандидата: Петров
Фамилия кандидата: Иванов
Фамилия кандидата: Иванов
Фамилия кандидата: Петров
Иванов => 3
Петров => 2
```

**Пример.** Создать частотный словарь. В словаре необходимо представить информацию о частоте вхождения в текст каждой буквы и пробела.

```
# Ввод текста
text = input('Введите текст: ')

# Создание и заполнение словаря
D = {}
for c in text:
    if c in D:
        D[c] = D[c] + 1
    else:
        D[c] = 1

# Вывод словаря
for k in D:
    print(k, D[k])
```

Введите текст: Карл у Клары украл кораллы

К 2  
а 4  
р 4  
л 5  
  4  
у 2  
ы 2  
к 2  
о 1

**Пример.** Вводится прописной эквивалент целого положительного числа (т.е. число, записанное словами): *два миллиона сто двадцать три тысячи восемьсот тридцать пять минус одна тысяча шестьсот*. Представить запись десятичным числом.

```

▼ unit_dict = dict(один=1, одна=1, два=2, две=2, три=3, четыре=4, пять=5,
                   шесть=6, семь=7, восемь=8, девять=9, десять=10,
                   одиннадцать=11, двенадцать=12, тринадцать=13,
                   четырнадцать=14, пятнадцать=15, шестнадцать=16,
                   семнадцать=17, восемнадцать=18, девятнадцать=19, двадцать=20,
                   тридцать=30, сорок=40, пятьдесят=50, шестьдесят=60,
                   семьдесят=70, восемьдесят=80, девяносто=90, сто=100,
                   двести=200, триста=300, четыреста=400, пятьсот=500,
                   шестьсот=600, семьсот=700, восемьсот=800, девятьсот=900)
▼ multiplier_dict = dict(тысяча=10**3, тысяч=10**3, тысячи=10**3, миллион=10**6,
                         миллиона=10**6, миллионов=10**6, миллиард=10**9,
                         миллиарда=10**9, миллиардов=10**9)

input_str = "сто два миллиона сто двадцать три тысячи восемьсот тридцать пять".lower()
# input_str = "восемьдесят два".Lower()
ls_str = input_str.split()
res, multiplier = 0, 1
▼ for word in ls_str[::-1]:
    if word in multiplier_dict:
        multiplier = multiplier_dict[word]
    if word in unit_dict:
        res += unit_dict[word] * multiplier
print(res)

```

82

#### 4.6.11 Контрольное задание. Использование словарей

**Задача.** Требуется смоделировать работу с телефонным справочником, в котором в качестве ключей записаны номера телефонов абонентов, содержащий также его фамилию, имя и дату рождения.

Необходимо программно заполнить справочник начальной информацией (чтобы меньше вводить данных при тестировании). Предусмотрите систему меню с пунктами:

- просмотр всех записей в базе данных;
- добавление  $N$  записей;
- удаление записи по ключу;
- изменение записи;
- поиск необходимой информации;
- завершение работы с базой данных.

Отображение пунктов меню можно реализовать в виде процедуры, например:

```

def menu():
    print()
    print(' МЕНЮ: '.center(40,'*'))
    print('1. Просмотр всех записей в базе данных')
    print('2. Добавление N записей')
    print('3. Удаление записи по ключу')
    print('4. Изменение записи')
    print('5. Поиск необходимой информации')
    print('6. Завершение работы с базой данных\n')
    num = int(input('Выберите пункт меню для продолжения: '))
    return num

```



```

▼ def menu():
    print()
    print(' МЕНЮ: '.center(40,'*'))
    print('1. Просмотр всех записей в базе данных')
    print('2. Добавление N записей')
    print('3. Удаление записи по ключу')
    print('4. Изменение записи')
    print('5. Поиск необходимой информации')
    print('6. Завершение работы с базой данных\n')
    num = int(input('Выберите пункт меню для продолжения: '))
    return num

▼ data = { '89105632512': ['Шарапов', 'Василий', '10.12.2001'],
            '89156214578': ['Кузьмин', 'Иннокентий', '12.05.1998'],
            '89201235896': ['Добрынин', 'Фёдор', '13.01.2000'],
            '89523215613': ['Сорокин', 'Егор', '25.05.1998'],
            '89156321547': ['Мартышкин', 'Макар', '16.06.1996'],
            '89109029755': ['Рожкин', 'Роман', '31.12.2002'],
            '89105632532': ['Маврина', 'Марина', '08.03.2003'],
            '89206548912': ['Дубинина', 'Евгения', '17.08.2000'],
            '89802364565': ['Сорокин', 'Назар', '15.07.1995'],
        }

num = 0
▼ while num != 6:
    num = menu()
    while num not in range(1, 7):
        num = int(input('Повторите выбор пункта меню: '))
    if num == 1: # Просмотр всех записей в базе данных
        print('Фамилия'.center(15), 'Имя'.center(12), 'Телефон'.center(17),
              'Дата рождения'.center(15))
        print('.center(15, '-'), '.center(12, '-'), '.center(17, '-'),
              '.center(15, '-'))
        for key, val in data.items():
            print('{0:<15s}'.format(val[0]), '{0:<12s}'.format(val[1]),
                  key.center(17), val[2].center(12))
    elif num == 2: # Добавление N записей
        n = int(input('Введите количество новых записей '))
        for _ in range(n):
            key = input(' Введите номер телефона: ')
            fam, name = input(' Введите фамилию и имя: ').split()
            dr = input(' Введите дату рождения: ')
            if key in data:
                print('Такой телефон записан на абоненте: ', data[key][0],
                      data[key][1])
                ans = input('Обновить запись по введенному номеру (д/н)? ')
                if ans.lower() == 'д':
                    data[key][0], data[key][1], data[key][2] = fam, name, dr
                else:
                    data[key] = [fam, name, dr]
    elif num == 3: # Удаление записи по ключу
        key = input('Введите номер телефона для удаления: ')
        if key in data:
            del data[key]
            print('Запись удалена!')
        else:
            print('Нет записи с указанным номером телефона')
    elif num == 4: # Изменение записи
        pass

```

```

    elif num == 5: # Поиск необходимой информации
        family = input('Введите фамилию для поиска номера телефона: ')
        find = False
        for key, val in data.items():
            if val[0] == family:
                find = True
                print(' %s %s - %s' % (val[0], val[1], key))
        if not find:
            print('Нет записи с такой фамилией!')
    else:
        print('Работа с программой завершена.')

```

\*\*\*\*\* МЕНЮ: \*\*\*\*\*

1. Просмотр всех записей в базе данных
2. Добавление  $N$  записей
3. Удаление записи по ключу
4. Изменение записи
5. Поиск необходимой информации
6. Завершение работы с базой данных

Выберите пункт меню для продолжения: 1

Фамилия	Имя	Телефон	Дата рождения
Шарапов	Василий	89105632512	10.12.2001
Кузьмин	Иннокентий	89156214578	12.05.1998
Добрынин	Фёдор	89201235896	13.01.2000
Сорокин	Егор	89523215613	25.05.1998
Мартышкин	Макар	89156321547	16.06.1996
Рожкин	Роман	89109029755	31.12.2002
Маврина	Марина	89105632532	08.03.2003
Дубинина	Евгения	89206548912	17.08.2000
Сорокин	Назар	89802364565	15.07.1995

\*\*\*\*\* МЕНЮ: \*\*\*\*\*

1. Просмотр всех записей в базе данных
2. Добавление  $N$  записей
3. Удаление записи по ключу
4. Изменение записи
5. Поиск необходимой информации
6. Завершение работы с базой данных

Выберите пункт меню для продолжения: 3

Введите номер телефона для удаления: 34535

Нет записи с указанным номером телефона

\*\*\*\*\* МЕНЮ: \*\*\*\*\*

1. Просмотр всех записей в базе данных
2. Добавление  $N$  записей
3. Удаление записи по ключу
4. Изменение записи
5. Поиск необходимой информации
6. Завершение работы с базой данных

Выберите пункт меню для продолжения: 3

Введите номер телефона для удаления: 89523215613

Запись удалена!

\*\*\*\*\* МЕНЮ: \*\*\*\*\*

1. Просмотр всех записей в базе данных
2. Добавление  $N$  записей
3. Удаление записи по ключу
4. Изменение записи
5. Поиск необходимой информации
6. Завершение работы с базой данных

Выберите пункт меню для продолжения: 1

Фамилия	Имя	Телефон	Дата рождения
Шарапов	Василий	89105632512	10.12.2001
Кузьмин	Иннокентий	89156214578	12.05.1998
Добрынин	Фёдор	89201235896	13.01.2000
Мартышкин	Макар	89156321547	16.06.1996
Рожкин	Роман	89109029755	31.12.2002
Маврина	Марина	89105632532	08.03.2003
Дубинина	Евгения	89206548912	17.08.2000
Сорокин	Назар	89802364565	15.07.1995

\*\*\*\*\* МЕНЮ: \*\*\*\*\*

1. Просмотр всех записей в базе данных
2. Добавление  $N$  записей
3. Удаление записи по ключу
4. Изменение записи
5. Поиск необходимой информации
6. Завершение работы с базой данных

Выберите пункт меню для продолжения: 6

Работа с программой завершена.

#### 4.6.12 Задания для самостоятельного выполнения

Во тех вариантах, где моделируется работа с базой данной, необходимо программно заполнить словарь информацией (чтобы меньше вводить данных при тестировании). Предусмотрите систему меню с пунктами:

- просмотр всех записей в базе данных;
- добавление  $N$  записей;
- удаление записи по ключу;
- поиск необходимой информации;
- завершение работы с базой данных.

Отображаемая информация должна быть представлена в таблице.

**Задача 1.** Составьте толковый словарь (информация по нескольким словам уже содержится в словаре). Дополните словарь некоторыми словами: сначала вводится слово, а затем через тире описание его значения. Все слова, значение которых записано в словаре, различны. В следующей строке записан запрос — это слово, значение которого нужно найти. Выведите описание его значения из словаря или фразу «Нет в словаре», если такого слова нет в словаре. Запрос должен повторяться циклически, пока на вопрос "Продолжить поиск?" пользователь не ответит "Нет". Поиск должен осуществляться без учета регистра символов.

**Задача 2.** Дан русский текст. Текст может состоять из любых символов, вам необходимо транслитерировать (то есть заменить все русские буквы на английские) только русские буквы, а остальные оставить на месте. Строчные буквы заменяются на строчные, заглавные заменяются на заглавные. Если заглавная буква превращается при транслитерации в несколько букв, то заглавной

должна оставаться только первая из них (например, «Ц» → «Сз»). Правила трансляции для больших букв: А-А, Б-В, В-В, Г-Г, Д-Д, Е-Е, Ё-Ё, Ж-Ж, З-З, И-И, Й-Й, К-К, Л-Л, М-М, Н-Н, О-О, П-Р, Р-Р, С-С, Т-Т, У-У, Ф-Ф, Х-Х, Ц-Цз, Ч-Чч, Ш-Шш, Щ-Щщ, Ъ-", Ы-Ы', Ъ-' , Э-Э' , Ю-Үү, Я-Я . Например, это Юность моя -> е'то Yunost' moya .

*Примечание.* Специальные правила замены предусмотренные стандартом: Буква "ц" заменяется на букву "с" если за ней следуют буквы е, і, у, ј. Во всех остальных случаях буква "ц" заменяется на последовательность "cz". Можно протестировать перевод на сайте: <https://www.translit.site/ru/type/gost-7.79-system-b> (<https://www.translit.site/ru/type/gost-7.79-system-b>)

**Задача 3.** Имеется база данных работников предприятия (информация уже содержится в словаре). Дополнить ее информацией по нескольким сотрудникам. Среди работников данного предприятия найти тех, чья заработка за месяц ниже средней по предприятию, а также распечатать список тех, кто проработал на предприятии более 10 лет, с указанием их фамилии, зарплаты, стажа работы и должности. Предусмотреть поиск информации о сотруднике по фамилии. Запрос должен повторяться циклически, пока на вопрос "Продолжить поиск?" пользователь не ответит "Нет".

**Задача 4.** Имеется база данных учеников класса (информация уже содержится в словаре). Дополнить ее информацией по некоторым учащимся. Вычислить средний балл учеников класса, если известны оценки каждого ученика по математике, русскому языку и физике. Распечатать по алфавиту фамилии учеников, имеющих средний балл выше среднего в классе. Предусмотреть поиск информации об учащемся по фамилии. Запрос должен повторяться циклически, пока на вопрос "Продолжить поиск?" пользователь не ответит "Нет".

**Задача 5.** Имеется база данных учеников класса (информация уже содержится в словаре). Дополнить ее информацией по некоторым учащимся. Напишите программу, которая поможет находить номера телефонов по фамилии. В первой строке задано одно целое число — количество номеров телефонов. В следующих строках заданы телефоны и фамилия, имя, отчество их владельцев через пробел. В следующей строке записан запрос — это фамилия человека, чей телефон нужно найти. Вывести номер телефона согласно запросу. Если в телефонной книге нет телефонов человека с таким именем, выведите в соответствующей строке «Нет в телефонной книге». Запрос должен повторяться циклически, пока на вопрос "Продолжить поиск?" пользователь не ответит "Нет".

**Задача 6.** Имеется база данных, содержащая информацию о выполнении плана: наименование изделия, единица измерения, план выпуска, фактически выпущено. Дополните существующую информацию полем "Отклонение от плана". Помимо стандартных манипуляций с записями, предусмотреть вывод сведений о невыполнении плана по всем видам продукции. Результат отсортируйте по убыванию количества единиц, показывающих отклонение от плана (для невыпущенных изделий).

**Задача 7.** Имеется база данных с результатами многоборья - всего 5 видов спорта, где у каждого спортсмена выставлен результат, показанный им в конкретном виде соревнований (например, плавание 50 м. - 27.35 сек; бег 3 км. - 8.45 мин, и т.д.). Определить для каждого спортсмена его место по каждому виду спорта, а также в общем зачете. Вывести трех призеров соревнований.

**Задача 8.** Имеется база данных с информацией о купленной партии товара: наименование, цена за единицу, количество купленного. Дополнить таблицу стоимостью каждого наименования товара, подсчитать стоимость всей партии. В алфавитном порядке вывести наименование тех видов товара, цена за единицу у которых меньше средней цены за единицу.

**Задача 9.** Имеется база данных со сведениями об учениках класса: фамилия, имя, пол, вес, рост. Определить среднюю массу мальчиков и средний рост девочек. Кто из учеников класса самый высокий.

**Задача 10.** Имеется база данных с результатами сдачи экзаменов: фамилия студента, номер зачетной книжки, оценки по четырем предметам. Определить предмет, по которому в результате сдачи экзаменационной сессии были показаны наилучшие результаты. Распечатать список отличников и неуспевающих.

#### 4.6.13 Дополнительные материалы

Для углубленного погружения в тему следует ознакомиться с типами `defaultdict`, `OrderedDict`, `Counter`, `ChainMap`, `UserDict` из встроенного модуля `collections`. В некоторых случаях они могут пригодиться, существенно сократив длину кода.

В [ ]:

