In [22]:

```
# Работа со словарями
#После списков словарь является самым гибким встроенным типом.
#Если список – это упорядоченная коллекция, то словарь – неупорядоченная. Основные особенно
#Доступ осуществляется по ключу, а не по индексу. По аналогии со списком, в словаре можно п
#в цикле по ключам.
#Значения словаря хранятся в неотсортированном порядке, более того, ключи могут храниться н
#они добавляются.
#По аналогии со списками, словарь может хранить вложенные словари. Словарь может хранить в
#объекты любого типа (heterogeneous). Ключ в словаре – immutable тип, может быть строкой, ц
#float либо кортежем, состоящим из указанных типов.
#Словари реализованы как хеш-таблицы с быстрым доступом.
#Словари, так же как и списки, хранят ссылки на объекты, а не сами объекты.
#Словарь (dictionary) – это ассоциативный массив или хеш.
#Это неупорядоченное множество пар ключ: значение с требованием уникальности ключей.
#Пара фигурных скобок {} создает пустой словарь. В отличие от последовательностей, доступ к
#производится по ключу, а не по индексу, ключ может быть любого типа, ключ не допускает изм
#Основные операции над словарем – сохранение с заданным ключом и извлечение по нему значени
#Также можно удалить пару key: value с помощью инструкции del.
#Метод keys() для словаря возвращает список всех используемых ключей в произвольном порядке
#для сортировки списка нужно применить метод sort().
#Для определения наличия определенного ключа есть метод has_key(), который в версии 3.0 уст
#вместо него есть оператор in.
#Добавление нового объекта в словарь не требует предварительных проверок: если ранее ключу
#некоторое значение,
#оно будет перезаписано.
#Отличием словаря от списков является
#Обычные списки (массивы) представляют собой набор пронумерованных элементов, то есть для о
#списка необходимо указать его номер. Номер элемента в списке однозначно идентифицирует сам
#данные по числовым номерам не всегда оказывается удобно. Например, маршруты поездов в Росс
#численно-буквенным кодом (число и одна буква), также численно-буквенным кодом идентифициру
#то есть для хранения информации о рейсах поездов или самолетов в качестве идентификатора у
#использовать не число,
#а текстовую строку.
#Структура данных, позволяющая идентифицировать ее элементы не по числовому индексу, а по п
#называется словарем или ассоциативным массивом.
# Словарь состоит из двух объектов:ключа и значения
#Ключ идентифицирует элемент словаря, значение является данными, которые соответствуют данн
#Значения ключей – уникальны, двух одинаковых ключей в словаре быть не может.
#В языке Питон ключом может быть произвольный неизменяемый тип данных: целые и действительн
#Ключом в словаре не может быть
#множество, но может быть элемент типа frozenset: специальный тип данных, являющийся аналог
#который нельзя изменять после создания.
#Значением элемента словаря может быть любой тип данных, в том числе и изменяемый.
# Рассмотрим пример словаря стран и их столиц
# Создадим пустой словать Capitals
Capitals = dict() # создание словаря с помощью dict() см. ниже
# Заполним его несколькими значениями, индексом является название страны, а значением – наз
Capitals['Russia'] = 'Москва'
Capitals['France'] = 'Вариж'
Capitals['USA'] = 'Вашингтон'
Capitals['Belarus'] = 'Минск'
print(Capitals) # вывод словаря
Countries = ['Russia', 'France', 'USA', 'Belarus', 'Kazahstan'] #знакомые Вам списки
print(Countries) # вывод словаря
for country in Countries:
    # Для каждой страны из списка проверим, есть ли она в словаре Capitals
    if country in Capitals:
        print('Столица страны ' + country + ': ' + Capitals[country])
    else:
```

```
        print('В базе нет страны с названием ' + country)
#Теперь отличие словаря от списка стало понятным
#Словари нужно использовать в следующих случаях:
# 1. Подсчет числа каких-то объектов. В этом случае нужно завести словарь, в котором ключам
#а значениями – их количество.
#2. Хранение каких-либо данных, связанных с объектом. Ключи – объекты, значения – связанные
#Например, если нужно по названию месяца определить его порядковый номер, то это можно сдел
#словаря Num['January'] = 1; Num['February'] = 2; ....
#3. Установка соответствия между объектами (например, “родитель–потомок”).
#Ключ – объект, значение – соответствующий ему объект.
#Если нужен обычный массив, но масимальное значение индекса элемента очень велико,
#и при этом будут использоваться не все возможные индексы (так называемый “разреженный масс
#то можно использовать ассоциативный массив для экономии памяти.
```

```
{'Russia': 'Москва', 'France': 'Вариж', 'USA': 'Вашингтон', 'Belarus': 'Минс
к'}
['Russia', 'France', 'USA', 'Belarus', 'Kazahstan']
Столица страны Russia: Москва
Столица страны France: Вариж
Столица страны USA: Вашингтон
Столица страны Belarus: Минск
В базе нет страны с названием Kazahstan
```

In [20]:

```
#СОЗДАНИЕ СЛОВАРЯ
#Способ № 1, с помощью лиетрала
d = {} # пустой словарь
print(d)
#Обычное выражение – оно удобно, если словарь статичен:
l = {'name': 'mel', 'age': 45}
print(l)

dic = {'vanya' : 23323223, 'smith' : 32232332} #создаем словарь
dic['fedya'] = 33332222
print(dic)
print(dic['smith'])
del dic['vanya']
print(dic)
print(dic.keys())
#проверка наличия ключа в словаре возвращает True  если ключ есть, False  если ключа нет
# устаревший способ dic.has_key('fedya'))
existKey ='fedya' in dic  # Актуальный способ
print(existKey)
existKey ='fed' in dic
print(existKey)
```

```
{}
{'name': 'mel', 'age': 45}
{'vanya': 23323223, 'smith': 32232332, 'fedya': 33332222}
32232332
{'smith': 32232332, 'fedya': 33332222}
dict_keys(['smith', 'fedya'])
True
False
```

In [23]:

```
#Способ № 2, с помощью функции dict
s = dict(short='dict', long='dictionary')
l = dict([(1, 1), (2, 4)])
print(s)
print (l)
#Использование dict, как и {} имеет свою специфику
Capitals = {'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington'} # хорошо использоб
print(Capitals)
Capitals = dict(Russia = 'Moscow', Belarus = 'Minsk', USA = 'Washington')# хорошо использоб
#можно создавать большие словари, если в качестве аргументов передавать уже готовые списки,
#которые могут быть получены не обязательно перечислением всех элементов, а любым другим сп
#по ходу исполнения программы.
print(Capitals)
Capitals = dict([("Russia", "Moscow"), ("Ukraine", "Kiev"), ("USA", "Washington")])# функци
#каждый элемент которого является кортежем
#из двух элементов: ключа и значения.
Capitals = dict(zip(["Russia", "Ukraine", "USA"], ["Moscow", "Kiev", "Washington"]))#сполья
#передаются два списка одинаковой длины: список ключей и
#список значений
print(Capitals)
```

```
{'short': 'dict', 'long': 'dictionary'}
{1: 1, 2: 4}
{'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington'}
{'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington'}
{'Russia': 'Moscow', 'Ukraine': 'Kiev', 'USA': 'Washington'}
```

In [8]:

```
#Способ № 3, с помощью метода fromkeys
d = dict.fromkeys(['a', 'b'])
print(d)
s = dict.fromkeys(['a', 'b'], 100)
print(s)
```

```
{'a': None, 'b': None}
{'a': 100, 'b': 100}
```

In [10]:

```
#Способ № 4, с помощью генераторов
l = {a: a ** 2 for a in range(7)}
print(l)
```

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

In [25]:

```
#Работа с элементами словаря
#Основная операция: получение значения элемента по ключу, записывается так же, как и для сп
#Если элемента с заданным ключом нет в словаре, то возникает исключение KeyError.
Capitals = {'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington'}
print (Capitals['Russia'])
Capitals['Andorra']#вызываем KeyError
```

```
Moscow

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-25-e644a84d7a7c> in <module>
      4 Capitals = {'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washingt
on'}
      5 print (Capitals['Russia'])
----> 6 Capitals['Andora']

KeyError: 'Andora'
```

In [31]:

```
#Другой способ определения значения по ключу – метод get: A.get(key). Если элемента с ключо
#то возвращается значение None. В форме записи с двумя аргументами A.get(key, val) метод во
#если элемент с ключом key отсутствует в словаре.
Capitals = {'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington'}
print (Capitals.get('Russia'))
print(Capitals.get('Andora'))#возвращаем None
print (Capitals.get('Russia',1))
print (Capitals.get('Andorra',1))# вернем 1 если значения в словаре нет
```

```
Moscow
None
Moscow
1
```

In [35]:

```
#Проверить принадлежность элемента словарю можно операциями in и not in, как и для множеств
Capitals = {'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington'}
if 'Russia' in Capitals:
    print('Элемент в словаре')
key = 'Belarus'
if key in Capitals:
    print('Элемент '+key+ ' в словаре')
key1 = 'Andorra'
if key1 not in Capitals:
    print('Элемент '+key1+ ' не в словаре')
```

```
Элемент в словаре
Элемент Belarus в словаре
Элемент Andora не в словаре
```

In [36]:

```
#Для добавления нового элемента в словарь нужно просто присвоить ему какое-то значение: A[k
Capitals = {'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington'}
print (Capitals)
Capitals['Andorra'] = 'Андорра-ла-Велла'
print (Capitals)
```

```
{'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington'}
{'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington', 'Andorra': 'Ан
дорра-ла-Велла'}
```

In [38]:

```
#Для удаления элемента из словаря можно использовать операцию del A[key] (операция возбужда
#если такого ключа в словаре нет.
Capitals = {'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington'}
print (Capitals)
key = 'USA'
del Capitals[key] # Удалили элемент с существующим заданным ключом
print (Capitals)
#попробуем удалить несуществующий элемент
key = 'Andorra'
del Capitals[key] # Удалили элемент с несуществующим заданным ключом
print (Capitals)
```

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

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-38-07cb68e9f5ec> in <module>
      7 #попробуем удалить несуществующий элемент
      8 key = 'Andorra'
----> 9 del Capitals[key] # Удалили элемент с несуществующим заданным ключом
     10 print (Capitals)
     11

KeyError: 'Andorra'
```

In [41]:

```
#Рассмотрим два безопасных способа удаления элемента из словаря.
#первый способ
Capitals = {'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington'}
print (Capitals)
key1 = 'Andorra'
if key1 in Capitals:# предварительно проверяем наличие ключа в словаре
    del Capitals[key1]
    print(Capitals)
#как видно из примера KeyError не наступил
```

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

In [43]:

```
#второй способ
Capitals = {'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington'}
print (Capitals)
key = 'Andorra'
try:
    del Capitals[key]
except KeyError:# перехватываем исключение (ошибку)
    print('Не существует элемента с заданным ключом "' + key + '" в словаре')
```

```
{'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington'}
Не существует элемента с заданным ключом "Andorra" в словаре
```

In [47]:

```
#Еще один способ удалить элемент из словаря: использование метода pop: A.pop(key).
#Этот метод возвращает значение удаляемого элемента, если элемент с данным ключом отсутству
#то возбуждается исключение. Если методу pop передать второй параметр, то если элемент в сл
#то метод pop возвратит значение этого параметра. Это позволяет проще всего организовать бе
#из словаря: A.pop(key, None).
Capitals = {'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington'}
print (Capitals)
Capitals.pop('USA')
print (Capitals)
Capitals.pop('Andorra', 'Элемент отсутствует')#вернем сообщение 'Элемент отсутствует' при с
#Capitals.pop('Andorra')# если элемент с заданным ключом отсутствует то возвращается KeyErr
```

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

Out[47]:

```
'Элемент отсутствует'
```

In [63]:

```
#перебор элементов в словаре
D = dict(zip('Belarus', list(range(6)))) #создадим словарь
print(D)
for key in D:
    print(key, D[key])
#помним, что ключ является уникальным, посмотрим что будет, если использовать строку с повт
D = dict(zip('Russia', list(range(6)))) #создадим словарь
print(D)
for key in D:
    print(key, D[key])
#как видно из примера Python убрал вовторяющийся элемент s, при этом значение 2 отсутствует
```

```
{'B': 0, 'e': 1, 'l': 2, 'a': 3, 'r': 4, 'u': 5}
B 0
e 1
l 2
a 3
r 4
u 5
{'R': 0, 'u': 1, 's': 3, 'i': 4, 'a': 5}
R 0
u 1
s 3
i 4
a 5
```

In [62]:

```
#помним, что ключ является уникальным, посмотрим что будет, если использовать строку с повт
D = dict(zip('Russia', list(range(6)))) #создадим словарь
print(D)
for key in D:
    print(key, D[key])
```

```
{'R': 0, 'u': 1, 's': 3, 'i': 4, 'a': 5}
R 0
u 1
s 3
i 4
a 5
```

In [67]:

```
#Следующие методы возвращают представления элементов словаря. Представления во многом похож
#если менять значения элементов словаря.
#метод keys возвращает представление ключей всех элементов
Capitals = {'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington'}
print(Capitals.keys())
#метод values возвращает представление всех значений
print(Capitals.values())
#метод items возвращает представление всех пар (кортежей) из ключей и значений.
print(Capitals.items())
```

```
dict_keys(['Russia', 'Belarus', 'USA'])
dict_values(['Moscow', 'Minsk', 'Washington'])
dict_items([('Russia', 'Moscow'), ('Belarus', 'Minsk'), ('USA', 'Washingto
n')])
```

In [13]:

```
#чтобы быстро проверить, есть ли значение val среди всех значений элементов словаря Capital
Capitals = {'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington'}
val = 'Томск'
if val in Capitals.values():
    print('Значение '+val+ ' присутствует в словаре')
else:
    print('Значения '+val+' нет в словаре')
val = 'Moscow'
if val in Capitals.values():
    print('Значение '+val+ ' присутствует в словаре')
else:
    print('Значения '+val+' нет в словаре')
# чтобы быстро проверить, есть ли ключ key среди всех значений элементов словаря Capitals м
Capitals = {'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington'}
key = 'Andorra'
if key in Capitals.keys():
    print('Ключ '+key+ ' присутствует в словаре')
else:
    print('Ключ '+key+' нет в словаре')
key = 'USA'
if key in Capitals.keys():
    print('Ключ '+key+ ' присутствует в словаре')
else:
    print('Ключ '+key+' нет в словаре')
#организовать цикл так, чтобы в переменной key был ключ элемента, а в переменной val, было
key1 = 'Belarus'
val1 = 'Minsk'
for key, val in Capitals.items():
    if key==key1 and val==val1:
        print('Ключ '+key+ ' со значением ' + val+' присутствует в словаре')
    else:
        print('Ключа '+key+ ' со значением ' + val+' нет в словаре')
print('Проверка окончена')
```

```
Значения Томск нет в словаре
Значение Moscow присутствует в словаре
Ключ Andorra нет в словаре
Ключ USA присутствует в словаре
Ключа Russia со значением Moscow нет в словаре
Ключ Belarus со значением Minsk присутствует в словаре
Ключа USA со значением Washington нет в словаре
Проверка окончена
```

In [28]:

```
# рассмотрим другие методы словарей
#dict.copy() - возвращает копию словаря.
Capitals = {'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington'}
print(Capitals)
CapitalsCopy=Capitals.copy()
print(CapitalsCopy)
#dict.clear() - очищает словарь.
CapitalsCopy.clear()
print(CapitalsCopy)
#dict.popitem() - удаляет и возвращает произвольную пару (ключ, значение). Если словарь пус
#Помните, что словари неупорядочены
CapitalsCopy=Capitals.copy()
print(CapitalsCopy)
CapitalsCopy.popitem()
print(CapitalsCopy)
```

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

In [35]:

```
#dict.setdefault(key[, default]) - возвращает значение ключа, но если его нет, не бросает и
#а создает ключ с значением default (по умолчанию None)
Capitals = {'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington'}
if Capitals.keys() =='Andorra':
    print('Ключ существует')
else:
    print('Ключ не существует')
    Capitals.setdefault('Andorra', 'Андорра-ла-Велла') #ключ не существует поэтому в словар
    print(Capitals)
Capitals.setdefault('Russia', 'Санкт-Петербург') #ключ существует, поэтому изменения не про
print(Capitals)
```

```
Ключ не существует
{'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington', 'Andorra': 'Ан
дорра-ла-Велла'}
{'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington', 'Andorra': 'Ан
дорра-ла-Велла'}
```

In [45]:

```
#dict.update([other]) - обновляет словарь, добавляя пары (ключ, значение) из other.
#Существующие ключи перезаписываются. Возвращает None (не новый словарь!)
#dict.update([other], **kwargs)
Capitals = {'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington'}
Capitals.update({'Andorra': 'Андорра-ла-Велла'}) # добавили новый элемент в словарь
print(Capitals)
print(Capitals.update({'Andorra': 'Андорра-ла-Велла'})) #вернули None так как ключ существу
Capitals.update({'Andorra': 'Новосибирск'})# изменили значение у элемента
print(Capitals)
```

```
{'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington', 'Andorra': 'Ан
дорра-ла-Велла'}
None
{'Russia': 'Moscow', 'Belarus': 'Minsk', 'USA': 'Washington', 'Andorra': 'Но
восибирск'}
```

In [2]:

```
#Метод fromkeys() позволяет создать словарь из списка, элементы которого становятся ключами
#Применять метод можно как классу dict, так и к его объектам
a = [1, 2, 3]
b = dict.fromkeys(a)
print(b)
c = dict.fromkeys(a, 10)
print(c)
```

```
{1: None, 2: None, 3: None}
{1: 10, 2: 10, 3: 10}
```

In [4]:

```
#Кортежи
#Кортеж (tuple) – это неизменяемая структура данных, которая по своему подобию очень похожа
#Как вы наверное знаете, а если нет, то, пожалуйста, ознакомьтесь с седьмым уроком, список
#Т.е. если у нас есть список a = [1, 2, 3] и мы хотим заменить второй элемент с 2 на 15, то
#напрямую обратившись к элементу списка. С кортежем мы не можем производить такие операции,
#Существует несколько причин, по которым стоит использовать кортежи вместо списков.
#Одна из них – это обезопасить данные от случайного изменения. Если мы получили откуда-то м
#и у нас есть желание поработать с ним, но при этом непосредственно менять данные мы не соб
#это как раз тот случай, когда кортежи придутся как нельзя кстати. Используя их в данной за
#мы дополнительно получаем сразу несколько бонусов – во-первых, это экономия места. Дело в
#в памяти занимают меньший объем по сравнению со списками.
spisok = [10, 20, 30]
korteg = (10, 20, 30)
print(spisok.__sizeof__())
print(korteg.__sizeof__())
#Во-вторых – прирост производительности, который связан с тем, что кортежи работают быстрее
#(т.е. на операции перебора элементов и т.п. будет тратиться меньше времени). Важно также о
#что кортежи можно использовать в качестве ключа у словаря
```

```
64
48
```

In [7]:

```
#Создание кортежей
a = ()
print(type(a))
b = tuple()
print(type(b))
#Кортеж с заданным содержанием создается также как список, только вместо квадратных скобок
c = (1, 2, 3, 4, 5)
print(type(c))
print(c)
#При желании можно воспользоваться функцией tuple()
d = tuple((1, 2, 3, 4))
print(d)
```

```
<class 'tuple'>
<class 'tuple'>
<class 'tuple'>
(1, 2, 3, 4, 5)
(1, 2, 3, 4)
```

In [8]:

```
#Доступ к элементам кортежа
a = (1, 2, 3, 4, 5)
print(a[0])
print(a[1:3])
#как мы помним изменять элементы кортежа нельзя, если есть желание посмотерть на результат
# приведенную ниже
#a[1] = 3
```

```
1
(2, 3)
```

In [9]:

```
#Удаление кортежей
#Удалить отдельные элементы из кортежа невозможно.
#уберите комментарй у строк ниже чтоб уюедиться в этом
#a = (1, 2, 3, 4, 5)
#del a[0]
#но можно удалить полностью кортеж
del a
print(a)
# в ошибке указано что объекту a не существует
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-9-0bff5ebaf6be> in <module>
      6 #но можно удалить полностью кортеж
      7 del a
----> 8 print(a)
      9 # в ошибке указано что объекту a не существует

NameError: name 'a' is not defined
```

In [11]:

```
#Кортеж можно преодразовать в список и обратно
lst = [1, 2, 3, 4, 5]
print(type(lst))
print(lst)
tpl = tuple(lst)
print(type(tpl))
print(tpl)
# обратное преобразование
tpl = (2, 4, 6, 8, 10)
print(type(tpl))
print(tpl)
lst = list(tpl)
print(type(lst))
print(lst)
```

```
<class 'list'>
[1, 2, 3, 4, 5]
<class 'tuple'>
(1, 2, 3, 4, 5)
<class 'tuple'>
(2, 4, 6, 8, 10)
<class 'list'>
[2, 4, 6, 8, 10]
```

In [13]:

```
#создать кортеж из итерируемого объекта можно с помощью все той же  функции tuple()
a = tuple('hello, world!')
print(a)
```

```
('h', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd', '!')
```

In [15]:

```
#Множества
#Множество в python - "контейнер", содержащий не повторяющиеся элементы в случайном порядке
#Создаём множества:
a = set()
print(a)
a = set('hello')
print(a)
a = {'a', 'b', 'c', 'd'}
print(a)
a = {i ** 2 for i in range(10)} # генератор множеств
a = {}  # А так нельзя, обратите внимание на класс созданного объекта!
print(type(a))
#Как видно из примера, множества имеет тот же литерал, что и словарь, но пустое множество с
```

```
set()
{'e', 'l', 'o', 'h'}
{'d', 'c', 'b', 'a'}
<class 'dict'>
```

In [17]:

```
#Множества удобно использовать для удаления повторяющихся элементов:
words = ['hello', 'daddy', 'hello', 'mum']
print(set(words))
```

```
{'hello', 'daddy', 'mum'}
```

In [37]:

```
#С множествами можно выполнять множество операций: находить объединение, пересечение...
a = {1, 2,3,4,5,6,7}
b = {5,6,7,8,9,10}
c = {80,9,90}
e = {6,7}
f = {7,6}
#len(s) - число элементов в множестве (размер множества).
print(len(a))
print(len(b))
#x in s - принадлежит ли x множеству s.
k=10
if k in a:
    print('Элемент к = '+ str(k) +' принадлежит множеству a')
else:
    print('Элемент к = '+ str(k) +' не принадлежит множеству a')
k=5
if k in b:
    print('Элемент к = '+ str(k) +' принадлежит множеству b')
else:
    print('Элемент к = '+ str(k) +' не принадлежит множеству b')

#set.isdisjoint(other) - истина, если set и other не имеют общих элементов.
print(a.isdisjoint(c))
print(a.isdisjoint(b))
#set == other - все элементы set принадлежат other, все элементы other принадлежат set.
if a==e:
    print('Все элементы a принадлежат e')
else:
    print('Не все элементы a принадлежат e')
if e==f:
    print('Все элементы e принадлежат f')
else:
    print('Не все элементы e принадлежат a')

#set.issubset(other) или set <= other - все элементы set принадлежат other.
if a<=e:
    print('Все элементы a принадлежат e')
else:
    print('Не все элементы a принадлежат e')
if e<=f:
    print('Все элементы e принадлежат f')
else:
    print('Не все элементы e принадлежат a')
#set.issuperset(other) или set >= other - аналогично.
if a>=e:
    print('Все элементы a принадлежат e')
else:
    print('Не все элементы a принадлежат e')
if e>=f:
    print('Все элементы e принадлежат f')
else:
    print('Не все элементы e принадлежат a')
#set.union(other, ...) или set | other | ... - объединение нескольких множеств.
t = a.union(b,c,e)
print(t)
#set.intersection(other, ...) или set & other & ... - пересечение.
y = a.intersection(b)
print(y)
#set.difference(other, ...) или set - other - ... - множество из всех элементов set, не при
```

```
i = a.intersection(b)
print(i)

#set.symmetric_difference(other); set ^ other - множество из элементов, встречающихся в одн
#но не встречающиеся в обоих.
j = a.intersection(b)
print(j)

#set.copy() - копия множества.
```

```
7
6
Элемент к = 10 не принадлежит множеству a
Элемент к = 5 принадлежит множеству b
True
False
Не все элементы a принадлежат e
Все элементы e принадлежат f
Не все элементы a принадлежат e
Все элементы e принадлежат f
Все элементы a принадлежат e
Все элементы e принадлежат f
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 80, 90}
{5, 6, 7}
{5, 6, 7}
{5, 6, 7}
```

In [61]:

```
a = {1, 2,3,4,5,6,7}
b = {5,6,7,8,9,10}
c = {80,90}
e = {1,2,3,4,5}
f = {3,4,5,6,7,8}
#set.update(other, ...); set |= other | ... - объединение.
l = e.update(f)
print(l)
#set.intersection_update(other, ...); set &= other & ... - пересечение.
k = e.intersection_update(f)
print(k)
#set.difference_update(other, ...); set -= other | ... - вычитание.
m = e.difference_update(f)
print(m)
#set.symmetric_difference_update(other); set ^= other - множество из элементов,
#встречающихся в одном множестве, но не встречающиеся в обоих.
#set.add(elem) - добавляет элемент в множество.
f.add(10)
print(f)
#set.remove(elem) - удаляет элемент из множества. KeyError, если такого элемента не существ
f.remove(10)
print(f)
#set.discard(elem) - удаляет элемент, если он находится в множестве.
f.discard(5)
print(f)
#set.pop() - удаляет первый элемент из множества. Так как множества не упорядочены,
#нельзя точно сказать, какой элемент будет первым.
f.pop()
print(f)
#set.clear() - очистка множества
f.clear()
print(f)
```

```
None
None
None
{3, 4, 5, 6, 7, 8, 10}
{3, 4, 5, 6, 7, 8}
{3, 4, 6, 7, 8}
{4, 6, 7, 8}
set()
```

In [65]:

```
#Отличие set от frozenset
a = set('qwerty')
b = frozenset('qwerty')
print(a == b)
print(type(a - b))
print(type(a | b))
print(a.add(1))
#строка ниже выдаст ошибку, угадайте почему
#print(b.add(1))
```

```
True
<class 'set'>
<class 'set'>
None
```

In [64]:

```
#Задания
#Используя знания в списках, множествах, кортежах и словарях:
#1. создать словарь адрессной книги, содержащий ФИО и адрес. Заполнить его 50 элементами. р
#2. создать словарь телефонного справочника. Заполнить его 50 элементами. Реализовать поиск
#3. реализовать проверку на существующие записи в предыдущих заданиях с возможностью дополн
#4. создать словарь на свободную тему, включающий в себя кортеж в качестве ключа. реализова
#5. создать словарь авиарейсов, в возможностью поиска маршрута из точки А в точку В с учето
#6. создать словарь железнодорожных сообщений с учетом более одной но менее 4 переседок, с
#времени поездки
```

In [ ]:

```

```