Лекция 3 "Словари, множества и выражения-генераторы"

часть 1 Словари и множества

Финансовый университет при Правительстве РФ, лектор С.В. Макрушин

v 0.6 02.10.2020

Разделы:

In [22]:

```
# загружаем стиль для оформления презентации
from IPython.display import HTML
from urllib.request import urlopen
html = urlopen("file:./lec_v1.css")
HTML(html.read().decode('utf-8'))
```

Out[22]:

Словари

- к оглавлению

Def: **Словари** - это наборы объектов, доступ к которым осуществляется не по индексу, а **по ключу**.

- В качестве **ключа** можно указать **неизменяемый объект**, например число, строку или кортеж.
- **Элементы словаря** могут содержать объекты **произвольного типа данных** и иметь неограниченную степень вложенности.
- Чтобы получить элемент, необходимо указать ключ, который использовался при сохранении значения.
- В словаре можно изменить элемент, хранящийся по определенному ключу.

Порядок эклментов в словаре:

- До Python 3.6 порядок хранения элементов в словаре не гарантировался. Т.е. обход элементов словаря мог привести к произвольному порядку получения элементов.
- Начиная с Python 3.6 в словарях порядок обхода стал соответствовать порядку, в котором элементы сохранялись в словаре. Однако, такое поведение являлось всего лишь спецификой реализации

референсной реализации Python CPython, а не спецификой языка Python.
- Начиная с Python 3.7 сохранение порядка вставки стало требованием языка и все реализации Python 3.7 должны реализовывать эту логику.

Создание словаря

- к оглавлению

Создание словаря с помощью конструктора `dict()`. Допустимые форматы вызова конструктора:
* `dict(<Ключ1>=<Значение1>[, ... , <КлючN>=<ЗначениеN>])`
* `dict(<Словарь>)`
* `dict(<Список кортежей с двумя элементами (Ключ, Значение)>)`
* `dict(<Список списков с двумя элементами [Ключ, Значение]>)`

In [2]:

```
dict() # пустой словарь
```

Out[2]:

```
{}
```

In [3]:

```
# при данном способе имена ключей должны быть корректными идентификаторами Python:
dict(a=1, b=2)
```

Out[3]:

```
{'a': 1, 'b': 2}
```

In [4]:

```
# ошибка:
dict(1a=1, b=2)
```

```
  File "<ipython-input-4-91c01e426dc6>", line 2
    dict(1a=1, b=2)
         ^
SyntaxError: invalid syntax
```

In [5]:

```
# создание словаря без явного использования конструктора:
d0 = {'a': 1, 'b': 2}
```

In [11]:

```
# создание словаря на основе другого словаря (копирование словаря):
d0c = dict(d0)
```

In [12]:

```
# ... на основе списка кортежей вида: (ключ, значение):
dict([('a', 1), ('b', 1)])
```

Out[12]:

```
{'a': 1, 'b': 1}
```

In [13]:

```
# ... на основе списка списков вида: (ключ, значение):
dict([['a', 1], ['b', 1]])
```

Out[13]:

```
{'a': 1, 'b': 1}
```

Объединить два списка в список кортежей позволяет функция `zip()`

In [14]:

```
k = ['a', 'b', 'c']
v = [1, 2, 3]
```

In [15]:

```
lkv = list(zip(k, v))
lkv
```

Out[15]:

```
[('a', 1), ('b', 2), ('c', 3)]
```

In [16]:

```
dict(lkv)
```

Out[16]:

```
{'a': 1, 'b': 2, 'c': 3}
```

В действительности конструктор `dict()` работает не только со списками, но и с итерируемыми объектами:

In [17]:

```
# dict можно применять напрямую к zip:
dict(zip(k, v))
```

Out[17]:

```
{'a': 1, 'b': 2, 'c': 3}
```

In [18]:

```
zip(k, v) # zip() возвращает не список, а итерируемый объект!
```

Out[18]:

```
<zip at 0x16f567ed6c8>
```

In [19]:

```
list(zip('hello', [1, 2])) # zip() заканчивает работу как только кончается один из итерируе
```

Out[19]:

```
[('h', 1), ('e', 2)]
```

Создание словаря при помощи указания всех элементов словаря внутри фигурных скобок. Это наиболее часто используемый способ создания словаря. Между ключом и значением указывается двоеточие, а пары "ключ/значение" перечисляются через запятую.

In [13]:

```
d1 = {} # пустой словарь
d1
```

Out[13]:

```
{}
```

In [14]:

```
d2 = {'a':1, 'b':2}
d2
```

Out[14]:

```
{'a': 1, 'b': 2}
```

In [28]:

```
d2 = {'1a':1, 'b':2}
d2
```

Out[28]:

```
{'1a': 1, 'b': 2}
```

Создание словаря при помощи поэлементного заполнения словаря.

In [29]:

```
d3 = {} # создание пустого словаря
d3['a'] = 1 # добавление элемента в словарь
d3['b'] = 2
d3
```

Out[29]:

```
{'a': 1, 'b': 2}
```

In [30]:

```
# напоминание: ключами словаря могут быть только неизменяемые объекты (кортежи, строки, чис
d3[d2] = 3
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-30-e5900dffbc9b> in <module>()
      1 # напоминание: ключами словаря могут быть только неизменяемые объекты
(кортежи, строки, числа и т.д.)
----> 2 d3[d2] = 3

TypeError: unhashable type: 'dict'
```

Создание словаря с помощью метода `dict.fromkeys(<Последовательность> [, <Значение>] )`. Метод создает новый словарь, ключами которого будут элементы последовательности. Если второй параметр не указан, то значением элементов словаря будет значение `None`.

In [16]:

```
d4 = dict.fromkeys(['a', 'b'], 1)
d4
```

Out[16]:

```
{'a': 1, 'b': 1}
```

In [17]:

```
d5 = dict.fromkeys(['a', 'b'])
d5
```

Out[17]:

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

Создание поверхностной копии с помощью функции `dict()`:

In [18]:

```
d6 = dict(d4)
d6
```

Out[18]:

```
{'a': 1, 'b': 1}
```

In [19]:

```
d6 is d4 # словари разные
```

Out[19]:

```
False
```

Создание поверхностной копии с помощью функции copy()

In [20]:

```
d7 = d6.copy()
d7
```

Out[20]:

```
{'a': 1, 'b': 1}
```

In [21]:

```
d7 is d6 # словари разные
```

Out[21]:

```
False
```

Создание глубокой копии словаря

In [22]:

```
import copy
d8 = copy.deepcopy(d6)
d8
```

Out[22]:

```
{'a': 1, 'b': 1}
```

>

Операции над словарями

- к оглавлению

In [31]:

```
d9 = {1: 'int', 'a': 'string', (1, 2): 'tup1e'}
```

In [32]:

```
d9[1]
```

Out[32]:

```
'int'
```

In [33]:

```
d9['a']
```

Out[33]:

```
'string'
```

In [34]:

```
d9[(1, 2)]
```

Out[34]:

```
'tup1e'
```

In [35]:

```
d9['c'] # ошибка! Обращение к несуществующему элементу
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-35-5030f182352d> in <module>()
----> 1 d9['c'] # ошибка! Обращение к несуществующему элементу

KeyError: 'c'
```

Проверить существование ключа можно с помощью оператора `in` :

- если ключ найден, то возвращается значение `True`
- в противном случае - `False` .

In [36]:

```
'a' in d9
```

Out[36]:

```
True
```

In [37]:

```
'c' in d9
```

Out[37]:

```
False
```

Определение размера словаря:

In [38]:

```
len(d9)
```

Out[38]:

```
3
```

Метод `get(<Ключ> [, <Значение по умолчанию>])` позволяет избежать вывода сообщения об ошибке при отсутствии указанного ключа:

- если ключ присутствует в словаре, то метод возвращает значение, соответствующее этому ключу
- если ключ отсутствует, то возвращается значение `None` или значение, указанное во втором параметре.

In [40]:

```
print(d9.get('a'))
```

```
string
```

In [41]:

```
print(d9.get('c')) # обращение к несуществующему элементу
```

```
None
```

In [42]:

```
d9.get('c', 'default')
```

Out[42]:

```
'default'
```

In [43]:

```
d9.get('a', 'default')
```

Out[43]:

```
'string'
```

In [44]:

```
d91 = dict(zip('abcd', range(4)))
d91
```

Out[44]:

```
{'a': 0, 'b': 1, 'c': 2, 'd': 3}
```

In [45]:

```
# способ, НЕ позволяющий работать с отсутствующими элементами:
d91['a'] = d91['a'] + 1
d91
```

Out[45]:

```
{'a': 1, 'b': 1, 'c': 2, 'd': 3}
```

In [46]:

```
# способ, НЕ позволяющий работать с отсутствующими элементами:
d91['e'] = d91['e'] + 1
d91
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-46-72f7254324e9> in <module>()
      1 # способ НЕ позволяющий работать с отсутствующими элементами:
----> 2 d91['e'] = d91['e'] + 1
      3 d91

KeyError: 'e'
```

In [48]:

```
# способ, позволяющий работать с отсутствующими элементами:
d91['a'] = d91.get('a', 0) + 1
d91
```

Out[48]:

```
{'a': 3, 'b': 1, 'c': 2, 'd': 3}
```

In [49]:

```
d91['e'] = d91.get('e', 0) + 1
d91
```

Out[49]:

```
{'a': 3, 'b': 1, 'c': 2, 'd': 3, 'e': 1}
```

Кроме того, можно воспользоваться методом `setdefault(<Ключ>[, <Значение по умолчанию>])`:

- если ключ присутствует в словаре, то метод возвращает значение, соответствующее этому ключу
- если ключ отсутствует, то вставляет новый элемент, со значением, указанным во втором параметре
  - если второй параметр не указан, значением нового элемента будет `None`.

In [50]:

```
d10 = dict(a=1, b=2)
d10
```

Out[50]:

```
{'a': 1, 'b': 2}
```

In [55]:

```
print(d10.setdefault('a'))
```

```
1
```

In [52]:

```
d10
```

Out[52]:

```
{'a': 1, 'b': 2}
```

In [53]:

```
print(d10.setdefault('c'))
```

```
None
```

In [54]:

```
d10
```

Out[54]:

```
{'a': 1, 'b': 2, 'c': None}
```

In [56]:

```
d10.setdefault('d', 3)
```

Out[56]:

```
3
```

In [57]:

```
d10
```

Out[57]:

```
{'a': 1, 'b': 2, 'c': None, 'd': 3}
```

Так как словари относятся к изменяемым типам данных, мы можем изменить элемент по ключу. Если элемент с указанным ключом отсутствует в словаре, то он будет добавлен в словарь.

In [58]:

```
d11 = dict(a=1, b=2)
d11
```

Out[58]:

```
{'a': 1, 'b': 2}
```

In [59]:

```
d11['a'] = 11 # изменение значения по ключу
d11
```

Out[59]:

```
{'a': 11, 'b': 2}
```

In [60]:

```
d11['c'] = 13 # добавление значения
d11
```

Out[60]:

```
{'a': 11, 'b': 2, 'c': 13}
```

Получить количество ключей в словаре позволяет функция `len()`:

In [61]:

```
len(d11)
```

Out[61]:

```
3
```

Удалить элемент из словаря можно с помощью оператора `del`:

In [62]:

```
d12 = dict(a=1, b=2)
del d12['b']
d12
```

Out[62]:

```
{'a': 1}
```

>

Перебор элементов словаря

- к оглавлению

In [64]:

```
d13 = dict(a=5, b=12, c=9, d=5)
```

In [73]:

```
# обход ключей словаря в цикле for:
for k in d13:
    print(f'key: {k}')
```

```
key: a
key: b
key: c
key: d
```

In [74]:

```
# НЕ эффективный обход ключей и значений словаря в цикле for:
for k in d13:
    print(f'key: {k}, value: { d13[k]}')
```

```
key: a, value: 5
key: b, value: 12
key: c, value: 9
key: d, value: 5
```

In [75]:

```
# обход ключей словаря в цикле for:
for k in d13.keys():
    print(f'key: {k}')
```

```
key: a
key: b
key: c
key: d
```

In [67]:

```
# обход упорядоченных ключей в цикле for:
for k in sorted(d13.keys()):
    print(f'key: {k}, value: { d13[k]}')
```

```
key: a, value: 5
key: b, value: 12
key: c, value: 9
key: d, value: 5
```

In [68]:

```
# получение списка ключей
list(d13)
```

Out[68]:

```
['a', 'b', 'c', 'd']
```

In [69]:

```
# получение списка ключей
list(d13.keys())
```

Out[69]:

```
['a', 'b', 'c', 'd']
```

In [70]:

```
# обход значений словаря в цикле for:
for v in d13.values():
    print(f'value: {v}')
# в словаре могут содержаться одинаковые значения!
```

```
value: 5
value: 12
value: 9
value: 5
```

In [71]:

```
# получение списка значений:
list(d13.values())
```

Out[71]:

```
[5, 12, 9, 5]
```

In [72]:

```
# обход пар ключ-значение в цикле for:
for k, v in d13.items():
    print(f'key: {k}, value: {v}')
```

```
key: a, value: 5
key: b, value: 12
key: c, value: 9
key: d, value: 5
```

In [76]:

```
# получение списка кортежей ключ-значение:
list(d13.items())
```

Out[76]:

```
[('a', 5), ('b', 12), ('c', 9), ('d', 5)]
```

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

Представление словаря - это итерируемый объект, доступный только для чтения и хранящий элементы, ключи или значения словаря в зависимости от того, какое представление было запрошено.

Между представлениями и обычными итерируемыми объектами есть два различия:

- если словарь, для которого было получено представление, изменяется, то представление будет отражать эти изменения.
- представления ключей и элементов поддерживают некоторые операции, свойственные множествам. Допустим, у нас имеются представление словаря `v` и множество или представление словаря `x`; для этой пары поддерживаются следующие операции:
  - `v & x # Пересечение`
  - `v | x # Объединение`
  - `v - x # Разность`
  - `v ^ x # Строгая дизъюнкция`

Как обходить словарь и при необходимости удалять из него элементы?

In [77]:

```
d13c = dict(d13)
d13c
```

Out[77]:

```
{'a': 5, 'b': 12, 'c': 9, 'd': 5}
```

In [78]:

```
# попытка реализации тривиального способа решения задачи:
for k, v in d13c.items():
    if k > 'b':
        del d13c[k] # НЕЛЬЗЯ удалять ключ-значение из словаря во время итерации по представ
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-78-5ad8d33038ad> in <module>()
      1 # попытка реализации тривиального способа решения задачи:
----> 2 for k, v in d13c.items():
      3     if k > 'b':
      4         del d13c[k] # НЕЛЬЗЯ удалять ключ-значение из словаря во врем
я итерации по представлению этого словаря!

RuntimeError: dictionary changed size during iteration
```

In [46]:

```
# решение проблемы удаления:
d13c = dict(d13)
for k, v in list(d13c.items()): # сначала итерируемся по представлению словарь.items() и со
    if k > 'b':
        del d13c[k] # во время удаления представление уже не используется, используется его
d13c
```

Out[46]:

```
{'a': 5, 'b': 12}
```

`pop(<Ключ> [, <Значение по умолчанию>])` - удаляет элемент с указанным ключом и возвращает его значение. Если ключ отсутствует, то возвращается значение из второго параметра. Если ключ отсутствует и второй параметр не указан, то возбуждается исключение KeyError.

In [79]:

```
d14 = dict(a=5, b=12, c=9, d=5)
```

In [80]:

```
d14.pop('a')
```

Out[80]:

```
5
```

In [81]:

```
d14
```

Out[81]:

```
{'b': 12, 'c': 9, 'd': 5}
```

`popitem()` - удаляет произвольный элемент и возвращает кортеж из ключа и значения. Если словарь пустой, возбуждается исключение `KeyError`.

In [83]:

```
d15 = dict(a=5, b=12, c=9, d=5)
```

In [84]:

```
d15.popitem()
```

Out[84]:

```
('d', 5)
```

In [85]:

```
d15
```

Out[85]:

```
{'a': 5, 'b': 12, 'c': 9}
```

`clear()` - удаляет все элементы словаря. Метод ничего не возвращает в качестве значения.

In [126]:

```
d15.clear()
d15
```

Out[126]:

```
{}
```

`update()` - добавляет элементы в словарь. Метод изменяет текущий словарь и ничего не возвращает.
Форматы метода:

- `update(<Ключ1>=<Значение1>[,  ... ,  <КлючN>=<ЗначениеN>])`
- `update(<Словарь>)`
- `update(<Список кортежей с двумя элементами>)`
- `update(<Список списков с  двумя элементами>)`

Если элемент с указанным ключом уже присутствует в словаре, то его значение будет перезаписано.

In [86]:

```
d16 = dict(a=5, b=12, c=9, d=5)
```

In [88]:

```
d16.update(c=3, d=4)
d16
```

Out[88]:

```
{'a': 5, 'b': 12, 'c': 3, 'd': 4}
```

>

Множества

- к оглавлению

Def: **Множество** - это неупорядоченная коллекция уникальных элементов, с которой можно сравнивать

другие элементы, чтобы определить, принадлежат ли они этому множеству. Множество может содержать только элементы неизменяемых типов, например числа, строки, кортежи. Объявить множество можно с помощью конструктора `set()` .

In [21]:

```
s1 = set()
s1
```

Out[21]:

```
set()
```

In [90]:

```
# преобразуем список в множество:
s2 = set ([1, 2, 3, 4, 5])
s2
```

Out[90]:

```
{1, 2, 3, 4, 5}
```

In [91]:

```
# преобразуем список в множество:
s3 = set ([1, 2, 3, 1, 2, 3])
s3
```

Out[91]:

```
{1, 2, 3}
```

In [95]:

```
# использование итерируемого объекта вместо списка:
s21 = set(range(5))
s21
```

Out[95]:

```
{0, 1, 2, 3, 4}
```

В Python 3 можно также создать множество, указав элементы внутри фигурных скобок. Обратите внимание на то, что при указании пустых фигурных скобок будет создан словарь, а не множество. Чтобы создать пустое множество, следует использовать функцию `set()` .

In [96]:

```
{1, 2, 3, 1, 2, 3}
```

Out[96]:

```
{1, 2, 3}
```

In [97]:

```
# обход элементов множества в цикле for:
for v in s2:
    print(v, end=' ')
```

```
1 2 3 4 5
```

In [98]:

```
len(s2)
```

Out[98]:

```
5
```

Пример: обход уникальных значений в списке (порядок обхода не сохраняется):

In [121]:

```
# создаем список с повторяющимися элементами:
import random
rl = []
for _ in range(15):
    rl.append(random.randint(1, 15))
rl
```

Out[121]:

```
[11, 2, 13, 2, 15, 15, 14, 11, 10, 13, 13, 4, 14, 1, 13]
```

In [122]:

```
# обход уникальных значений в списке, порядок обхода не сохраняется:
for e in set(rl):
    print(e)
```

```
1
2
4
10
11
13
14
15
```

In [99]:

```
s3 = {1, 2, 3, 4}
s4 = {2, 4, 5, 6}
```

In [100]:

```
# объединение множеств:
s3 | s4
```

Out[100]:

```
{1, 2, 3, 4, 5, 6}
```

In [101]:

```
# объединение множеств:
s3.union(s4)
```

Out[101]:

```
{1, 2, 3, 4, 5, 6}
```

In [102]:

```
# добавляют элементы множества s4 во множество s5:
s5 = {7, 8 , 9}
s5.update(s4) # эквивалентно: s5 |= s4
s5
```

Out[102]:

```
{2, 4, 5, 6, 7, 8, 9}
```

In [103]:

```
s51 = s5.copy()
s51 |= s4
s51
```

Out[103]:

```
{2, 4, 5, 6, 7, 8, 9}
```

In [104]:

```
# & и intersection() - пересечение множеств:
s4 & s3 #  s4.intersection(s3)
```

Out[104]:

```
{2, 4}
```

`a &= b` и `a.intersection_update(b)` - во множестве a останутся элементы, которые существуют и во множестве a, и во множестве b

In [105]:

```
s7 = {2, 4, 5, 6}
```

In [106]:

```
s7 &= s3 # s7.intersection_update(s7)
s7
```

Out[106]:

```
{2, 4}
```

In [107]:

```
# разница множеств:
s4 - s3 # s4.difference(s3)
```

Out[107]:

```
{5, 6}
```

In [108]:

```
# удаляем элементы из множества a, которые присутствуют во множестве b:
s6 = {2, 4, 5, 6}
s6 -= s3 # s6.difference_update(s3)
s6
```

Out[108]:

```
{5, 6}
```

`^` и `symmetric_difference()` - возвращают элементы обоих множеств, присутствующие только в одном из множеств-аргументов.

In [109]:

```
{3, 4, 5, 6} ^ {5, 6, 7, 8}
```

Out[109]:

```
{3, 4, 7, 8}
```

`a ^= b` и `a.symmetric_difference_update(b)` - во множестве a будут все элементы обоих множеств, исключая одинаковые элементы.

Операторы сравнения множеств

`in` - проверка наличия элемента во множестве:

In [110]:

```
s3
```

Out[110]:

```
{1, 2, 3, 4}
```

In [111]:

```
2 in s3
```

Out[111]:

```
True
```

In [112]:

```
7 in s3
```

Out[112]:

```
False
```

== - проверка на равенство (совпадение значений множеств):

In [113]:

```
set([1, 2, 3]) == set([1, 2, 3])
```

Out[113]:

```
True
```

- `a <= b` и `a.issubset(b)` - проверяют, входят ли все элементы множества a во множество b
- `a < b` - проверяет, входят ли все элементы множества a во множество b. Причем множество a не должно быть равно множеству b
- `a >= b` и `a.issuperset(b)` - проверяют, входят ли все элементы множества b во множество a
- `a > b` - проверяет, входят ли все элементы множества b во множество a. Причем множество a не должно быть равно множеству b
- `a.isdisjoint(b)` - возвращает `True` , если результатом пересечения множеств a и b является пустое множество (это означает, что множества не имеют одинаковых элементов)

`copy()` - создает копию множества. Обратите внимание на то, что оператор `=` присваивает лишь ссылку на тот же объект, а не копирует его.

`add(<Элемент>)` - добавляет <Элемент> во множество:

In [154]:

```
s8 = {1, 2, 3}
s8.add(4)
s8
```

Out[154]:

```
{1, 2, 3, 4}
```

`remove(<Элемент>)` - удаляет <Элемент> из множества. Если элемент не найден, то возбуждается исключение `KeyError`

In [114]:

```
s9 = {1, 2, 3}
s9.remove(2)
s9
```

Out[114]:

```
{1, 3}
```

In [115]:

```
s9.remove(4)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-115-5681cab85515> in <module>()
----> 1 s9.remove(4)

KeyError: 4
```

`discard(<Элемент>)` - удаляет <Элемент> из множества, если он присутствует

In [116]:

```
s10 = {1, 2, 3}
s10.discard(2)
s10
```

Out[116]:

```
{1, 3}
```

In [160]:

```
s10.discard(4)
```

`pop()` - удаляет произвольный элемент из множества и возвращает его. Если элементов нет, то возбуждается исключение `KeyError`

In [117]:

```
s11 = {1, 2, 3}
s11.pop()
```

Out[117]:

```
1
```

In [118]:

```
s11
```

Out[118]:

```
{2, 3}
```

`clear()` - удаляет все элементы из множества

In [163]:

```
s11.clear()
s11
```

Out[163]:

```
set()
```

Задание к следующему разделу

По книге Н. Прохоренок:

Глава 11 Пользовательские функции повторить из Глав 8 и 9 разделы "Генераторы списков и выражения-генераторы" и "Генераторы словарей" и "Генераторы множеств"

По книге М. Саммерфильд:

Глава 4 Управляющие структуры и функции (разедел "Собственные функции") Глава 3 Типы коллекций (раздел "Генераторы списков", "Генераторы множеств", "Генераторы словарей")

In [ ]:

```

```