Словари (dict) в Python

Как выглядит словарь python

Что такое словарь.
Словарь – неупорядоченная структура данных, которая позволяет хранить пары «ключ – значение».
Вот пример словаря на Python:

```
dictionary = {'персона': 'человек',
              'марафон': 'гонка бегунов длиной около 26 миль',
              'противостоять': 'оставаться сильным, несмотря на
давление',
              'бежать': 'двигаться со скоростью'}
```

Данный словарь использует строки в качестве ключей, однако ключом может являться в принципе любой неизменяемый тип данных. Значением же конкретного ключа может быть что угодно.

Вот ещё один пример словаря, где ключами являются числа, а значениями – строки:

```
gender_dict = {0: 'муж',
               1: 'жен'}
story_count = {'сто': 100,
               'девяносто': 90,
               'двенадцать': 12,
               'пять': 5}
```

Особенности словаря

Важное уточнение: если вы попробуете использовать изменяемый тип данных в качестве ключа, то получите ошибку:

```
dictionary = {(1, 2.0): 'кортежи могут быть ключами',
              1: 'целые числа могут быть ключами',
              'бежать': 'строки тоже',
              ['носок', 1, 2.0]: 'а списки не могут'}
```

Прим. перев. На самом деле проблема не с изменяемыми, а с нехэшируемыми типами данных, но обычно это одно и то же.

Еще момент.
Числовые ключи в словарях подчиняются правилам сравнения чисел. Таким образом, `int(1)` и `float(1.0)` считаются одинаковым ключом, к стати `True` будет им же. Однако из-за того, что значения типа `float` сохраняются приближенно, **не рекомендуется** использовать их в качестве ключей.

```
my_dict = {True: 'yes', 1: 'no', 1.0: 'maybe'}
# {True: 'maybe'}
```

Работа со словарями

Создание словаря

С помощью литерала:

```
d = {}
# {}

d = {'dict_key': 1, 'dictionary': 2}
# {'dict_key': 1, 'dictionary': 2}
```

С помощью функции dict

```
d = dict(short='dict', long='dictionary')
# {'short': 'dict', 'long': 'dictionary'}

d = dict([(1, 1), (2, 4)])
# {1: 1, 2: 4}

d_str = dict(("ab", "bc"))
# {"a": "b", "b": "c"}
```

С помощью метода fromkeys

```
d = dict.fromkeys(['a', 'b'])
# {'a': None, 'b': None}
```

```
d = dict.fromkeys(['a', 'b'], 100)
# {'a': 100, 'b': 100}
```

С помощью упаковщика zip

```
key_list = ['marvel_hero', 'dc_hero']
value_list = ['Spiderman', 'Flash']
superhero_dict = dict(zip(key_list, value_list))

print((superhero_dict))
# {'marvel_hero': 'Spiderman', 'dc_hero': 'Flash'}
```

С помощью генераторов словарей

Генераторы словарей. Они имеют похожий на генераторы списков синтаксис, но возвращают словарь:

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

Подробнее про генераторы словарей можно прочитать тут

Но принцип довольно простой

```
# Явно
{ key:value for item in list if conditional }

# функцией dict
dict((key, value) for item in list if condition)
```

Также генератор удобен, когда нужно инициализировать какой-то имеющийся список ключей:

```
list_of_keys = ['q', 'w', 'e', 'r', 't']
generated_dict = {k: 0 for k in list_of_keys}

print(generated_dict)
# {'q': 0, 'w': 0, 'e': 0, 'r': 0, 't': 0}
```

Получение данных из словаря

Для получения значения конкретного ключа используются квадратные скобки `[]`.
Предположим, что в нашем словаре есть пара `'марафон': 26`.

```
dictionary = {'марафон': 26}

# берём значение с ключом "марафон"
dictionary['марафон']
# 26
```

Опять же, вы получите ошибку, если попытаетесь получить значение по несуществующему ключу.
Для избежания подобных ошибок существует метод `get`.

Добавление и обновление ключей

Добавление новых пар в словарь происходит достаточно просто:

```
# Добавляем ключ "туфля" со значением "род обуви, закрывающей ногу не выше щиколотки"
dictionary['туфля'] = 'род обуви, закрывающей ногу не выше щиколотки'
```

Обновление существующих значений происходит абсолютно также:

```
# Обновляем ключ "туфля" и присваиваем ему значение "хорошая туфля"
dictionary['туфля'] = 'хорошая туфля'
```

Удаление ключей

Для удаления ключа и соответствующего значения из словаря можно использовать del

```
# Удаляем значение с ключом "противостоять" из словаря
del dictionary['противостоять']
```

Методы словарей

Словари в Python имеют множество различных полезных методов, которые помогут вам в работе с ними. Вот лишь некоторые из них:

dict.clear()

Метод `clear` позволяет очистить словарь:

```
london = {'name': 'London1', 'location': 'London Str'}

london.clear()
# {}
```

dict.copy()

Метод `copy` позволяет создать полную копию словаря.

Если указать, что один словарь равен другому:

```
london = {'name': 'London1', 'location': 'London Str', 'vendor':
'Cisco'}
london2 = london
id(london)
# 25489072

id(london2)
# 25489072

london['vendor'] = 'Juniper'

london2['vendor']
# 'Juniper'
```

В этом случае london2 это еще одно имя, которое ссылается на словарь. И при изменениях словаря london меняется и словарь london2, так как это ссылки на один и тот же объект.

Поэтому, если нужно сделать копию словаря, надо использовать метод copy():

```
london = {'name': 'London1', 'location': 'London Str', 'vendor':
'Cisco'}
london2 = london.copy()
```

```
id(london)
# 25524512

id(london2)
# 25563296

london['vendor'] = 'Juniper'

london2['vendor']
# 'Cisco'
```

dict.get()

`dict.get(key, default)` - возвращает значение ключа, но если его нет, не бросает исключение, а возвращает `default` (по умолчанию `None`).

```
# Допустим, у нас есть словарь story_count
story_count = {'сто': 100,
               'девяносто': 90,
               'двенадцать': 12,
               'пять': 5}
```

Метод `get()` возвращает значение по указанному ключу. Если указанного ключа не существует, метод вернёт `None`.

```
# Ключ "двенадцать" существует и метод get в данном случае вернёт 12
story_count.get('двенадцать')
# 12
```

Метод можно использовать для проверки наличия ключей в словаре:

```
story_count.get('два')
# None
```

Также можно указать значение по умолчанию, которое будет возвращено вместо `None`, если ключа в словаре не окажется:

```
# Метод вернёт 0 в случае, если данного ключа не существует
```

```
story_count.get('два', 0)
```

dict.update()

Метод `update()` пригодится, если нужно обновить несколько пар сразу.
Метод принимает другой словарь в качестве аргумента.

```
# Начальный словарь
dictionary = {'персона': 'человек',
              'марафон': 'гонка бегунов длиной около 26 миль',
              'противостоять': 'оставаться сильным, несмотря на
давление',
              'бежать': 'двигаться со скоростью'}

# Добавляем две пары в словарь dictionary, используя метод update
dictionary.update({'бежал': 'бежать в прошедшем времени',
                   'туфли': 'туфля во множественном числе'})

print(dictionary)

#        {'марафон': 'гонка бегунов длиной около 26 миль',
#        'персона': 'человек',
#        'бежал': 'бежать в прошедшем времени',
#        'бежать': 'двигаться со скоростью',
#        'туфля': 'род обуви, закрывающей ногу не выше щиколотки',
#        'туфли': 'туфля во множественном числе'}
```

Если вас интересует, почему данные в словаре расположены не в том порядке, в котором они были внесены в него, то это потому что словари не упорядочены.

dict.pop()

Метод `pop()` удаляет ключ и возвращает соответствующее ему значение.

```
story_count = {'сто': 100,
               'девяносто': 90,
               'двенадцать': 12,
               'пять': 5}

story_count.pop('девяносто')
# >>> 90
```

```
print(story_count)
# >>> {'двенадцать': 12, 'сто': 100, 'пять': 5}
```

dict.setdefault()

Метод `setdefault` ищет ключ, и если его нет, вместо ошибки создает ключ со значением `None`.

```
london = {'name': 'London1', 'location': 'London Str', 'vendor':
'Cisco'}

ios = london.setdefault('ios')

print(ios)
# None

print(london)
# {'name': 'London1', 'location': 'London Str', 'vendor': 'Cisco',
'ios': None}
```

Если ключ есть, setdefault возвращает значение, которое ему соответствует:

```
london.setdefault('name')
# 'London1'
```

Второй аргумент позволяет указать, какое значение должно соответствовать ключу:

```
    In [26]: model = london.setdefault('model', 'Cisco3580')

    In [27]: print(model)
    Cisco3580

    In [28]: london
    Out[28]:
    {'name': 'London1',
     'location': 'London Str',
     'vendor': 'Cisco',
     'ios': None,
     'model': 'Cisco3580'}
```

Метод `setdefault` заменяет такую конструкцию:

```
if key in london:
     value = london[key]
 else:
     london[key] = 'somevalue'
     value = london[key]
```

dict.keys()

Метод `keys()` возвращает коллекцию ключей в словаре.

```
story_count = {'сто': 100, 'девяносто': 90, 'двенадцать': 12, 'пять': 5}
print(story_count.keys())

# dict_keys(['сто', 'девяносто', 'двенадцать', 'пять'])
```

dict.values()

Метод `values()` возвращает коллекцию значений в словаре.

```
story_count = {'сто': 100, 'девяносто': 90, 'двенадцать': 12, 'пять': 5}
print(story_count.values())

# dict_values([100, 90, 12, 5])
```

dict.items()

Метод `items()` возвращает пары «ключ — значение».

```
story_count = {'сто': 100, 'девяносто': 90, 'двенадцать': 12, 'пять': 5}
story_count.items()

# dict_items([('сто', 100), ('девяносто', 90), ('двенадцать', 12),
('пять', 5)])
```

Примеры

Итерация через словарь

Вы можете провести итерацию по каждому ключу в словаре.

```
story_count = {'сто': 100, 'девяносто': 90, 'двенадцать': 12, 'пять': 5}
for key in story_count:
   print(key)

# сто
# девяносто
# двенадцать
# пять
```

Очевидно, вместо `story_count` можно использовать `story_count.keys()`.

В примере кода ниже ниже цикл `for` использует метод `items()` для получения пары «ключ – значение» на каждую итерацию.

```
for key, value in dictionary.items():
    print(key, value)

# >>> ('персона', 'человек')
# >>> ('бежать', 'двигаться со скоростью')
# >>> ('туфля', 'род обуви, закрывающей ногу не выше щиколотки')
# >>> ('бежал', 'бежать в прошедшем времени')
# >>> ('марафон', 'гонка бегунов длиной около 26 миль')
# >>> ('туфли', 'туфля во множественном числе')
```

Сортировка словаря

Сотритовка может производиться разными способами. Пример сортировки по ключу с помощью генератора

```
my_dict = {'сто': 100, 'девяносто': 90, 'двенадцать': 12, 'пять': 5}
result = {key: val for key, val in sorted(my_dict.items(), key = lambda
ele: ele[0])}

# result -> {'двенадцать': 12, 'девяносто': 90, 'пять': 5, 'сто': 100}
```

dict to list

Для конвертации dict в list достаточно проитерировать словарь попарно с помощью метода items(), и, на каждой итерации, добавлять пару ключ:значение к заранее

созданному списку. На выходе получим список списков, где каждый подсписок есть пара из исходного словаря.

```
medicine_chest = dict(top_part='potion', bot_part='bandage')
medicine_list = []
for key, con in medicine_chest.items():
    temp = [key, con]
    medicine_list.append(temp)

# [['top_part', 'potion'], ['bot_part', 'bandage']]
```

Тот же результат в одну строчку

```
my_dict = {'сто': 100, 'девяносто': 90, 'двенадцать': 12, 'пять': 5}
nl = [[k, v] for k, v in my_dict.items()]

# [['сто', 100], ['девяносто', 90], ['двенадцать', 12], ['пять', 5]]
```

Теги: python programing types dict

✎ Отредактировать эту страницу

*Последнее обновление **4 сент. 2023 г.** от **Stavis***