Программирование в лингвистике. Лекция 8

Словари. Методы словарей

Словари в Python

```
dct = {1: “a”, 2: “b”, 3: “c”}
```

* У словаря есть ключи и значения. Здесь 1, 2 и 3 – ключи, а ‘a’, ‘b’, и ‘c’ – значения.
* Ключи работают, как индексы списка. Как элементы множества, ключи могут быть только неизменяемыми, уникальными и неупорядоченными.

Так, в ключах могут быть числа, строки, кортежи, bool и даже None.

Словари в Python - неупорядоченные коллекции пар объектов с доступом по ключу. Их иногда ещё называют ассоциативными массивами или хеш-таблицами.

Значения словаря могут быть любыми. Они могут включать в себя другие словари, списки, множества и т.д.
Словари итерируемые и изменяемые.

Как создать словарь?

Способ 1: явно в коде.

```
dct = {"sky": "небо", "earth": "земля"}
```

Способ 2: преобразованием в словарь списка кортежей.

```
list_of_tuples = [("sky", "небо"), ("earth", "земля")]
dct = dict(list_of_tuples)
dct
```

```
{'sky': 'небо', 'earth': 'земля'}
```

Как создать словарь?

Способ 3: преобразованием объектов, возвращаемых функциями zip(list1, list2) и enumerate(list) (они возвращают списки кортежей)

zip()

```
lst1 = ["sky", "earth"]
lst2 = ["небо", "земля"]
dct = dict(zip(lst1, lst2))
dct
```

```
{'sky': 'небо', 'earth': 'земля'}
```

enumerate()

```
lst = ["zero", "one", "two"]
dct = dict(enumerate(lst))
dct
```

```
{0: 'zero', 1: 'one', 2: 'two'}
```

Как создать словарь?

Способ 4: задать ключи и их дефолтное значение с помощью dict.fromkeys([key1, key2, ...], default_value)

```
dct = dict.fromkeys(['one', 'two', 'three'], 0)
dct
```

```
{'one': 0, 'two': 0, 'three': 0}
```

Как создать словарь?

Способ 5: с помощью dict comprehension (генератор словарей).

```
squares = {x: x ** 2 for x in range(2, 6)}
squares
```

```
{2: 4, 3: 9, 4: 16, 5: 25}
```

Добавление элемента в словарь

Добавить элемент можно просто операцией присваивания – dict[key] = value.

```
dct = {"one": 1, "two": 2}
dct["three"] = 3
dct
```

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

Если такого ключа нет – ничего страшного. Однако, любая операция с отсутствующим ключом приведет к KeyError.

Удаление элемента

Способы 1 и 2. С помощью .pop() и .popitem()

```
dct = {"one": 1, "two": 2, "three": 3}
value = dct.pop("one")
print(value)
```

```
1
```

.pop() возвращает значение удаленного ключа.

```
key_value = dct.popitem()
key_value
```

```
('three', 3)
```

.popitem() удаляет последний добавленный элемент и возвращает пару (ключ, значение)

Удаление элемента

Способ 3. С помощью del.

```
dct = {"one": 1, "two": 2, "three": 3}
del dct["one"]
dct
```

```
{'two': 2, 'three': 3}
```

Обращение к значению по ключу

Получение значения по ключу – просто и удобно.
Нотация напоминает обращение к элементу списка по индексу.

```
dct = {"sky": "небо", "earth": "земля"}
print(dct["sky"])
print(dct["earth"])
```

```
небо
земля
```

Обращение к значению по ключу

Метод .get() позволяет сделать то же самое, только страхует нас в том случае, если такого ключа нет в словаре.

```
dct = {"sky": "небо", "earth": "земля"}
print(dct.get("grass"))
print(dct["grass"])
```

```
None
-----------
KeyError
```

```
print(dct.get("sky"))
```

```
небо
```

<- Обращение по несуществующему ключу вызывает KeyError. А .get() возвращает None.

View objects и словари

Итак, словарь состоит из ключей и значений. В Python существуют «взгляды» на объекты – view objects.

```
dct = {"sky": "небо", "earth": "земля"}
print(dct.keys())
print(dct.values())
print(dct.items())
```

```
dict_keys(['sky', 'earth'])
dict_values(['небо', 'земля'])
dict_items([('sky', 'небо'), ('earth', 'земля')])
```

Мы можем просмотреть часть словаря – ключи или значения, а также пары (ключ, значение). Мы не можем изменять их, но можем обратиться к ним.

Например, мы можем итерироваться по ключам, значениям или парам.

Итерируемся по словарю

Только по ключам:

```
dct = {"one": 1, "two": 2, "three": 3}
for key in dct:
  print(f'Ключ: {key}, значение: {dct[key]}')
```

```
Ключ: one, значение: 1
Ключ: two, значение: 2
Ключ: three, значение: 3
```

Итерируемся по словарю

То же самое:

```
dct = {"one": 1, "two": 2, "three": 3}
for key in dct.keys():
  print(f'Ключ: {key}, значение: {dct[key]}')
```

```
Ключ: one, значение: 1
Ключ: two, значение: 2
Ключ: three, значение: 3
```

Итерируемся по словарю

Только по значениям:

```
dct = {"one": 1, "two": 2, "three": 3}
for value in dct.values():
  print(f'Значение: {value}')
```

```
Значение: 1
Значение: 2
Значение: 3
```

Итерируемся по словарю

По парам - кортежам (ключ, значение):

```
dct = {"one": 1, "two": 2, "three": 3}
for key, value in dct.items():
  print(f'Ключ: {key}, значение: {value}')
```

```
Ключ: one, значение: 1
Ключ: two, значение: 2
Ключ: three, значение: 3
```

Пример

```
monthslist = ['January', 'February', 'March', 'April',
              'May', 'June', 'July', 'August', 'September',
              'October', 'November', 'December']
months = dict(enumerate(monthslist))
print('Что-то не очень, да?', months)
```

```
Что-то не очень, да? {0: 'January', 1: 'February', 2: 'March',
```

Здесь enumerate() как всегда подставит каждому элементу значение из range(0, len(list) + 1). Нам это не подходит.

Пример

```
months = {x + 1: y for x, y in enumerate(monthslist)}
print('Вот так уже лучше...')
for key, value in months.items():
  print(f'{value} is number {key}')
```

```
Вот так уже лучше...
January is number 1
February is number 2
March is number 3
April is number 4
```

Здесь мы искусственно увеличиваем каждый номер enumerate() на 1.

Пример

```
months = dict(enumerate(monthslist, 1))
print('Вот теперь совсем хорошо')
for key, value in months.items():
  print(f'{value} is number {key}')
```

```
Вот теперь совсем хорошо
January is number 1
February is number 2
March is number 3
April is number 4
May is number 5
June is number 6
July is number 7
August is number 8
```

Здесь мы передаем в enumerate аргумент – число, с которого начинается нумерация

Длина словаря

К словарю применима функция len().

```
dct = {"one": 1, "two": 2, "three": 3}
len(dct)
```

```
3
```

Наличие ключа

Наличие ключа проверяется с помощью in:

```
dct = {"one": 1, "two": 2, "three": 3}
print('one' in dct)
print('one' not in dct)
```

```
True
False
```

Соединение словарей

```
dct1 = {"a": 1, "b": 2}
dct2 = {"b": 3, "c": 4}
dct1.update(dct2)
dct1
```

```
{'a': 1, 'b': 3, 'c': 4}
```

In-place метод .update соединяет два словаря.
Если во втором словаре содержатся ключи, которые есть в первом, то значение ключа в первом словаре перезапишется значением второго.

Соединение словарей

```
dct1 = {"a": 1, "b": 2}
dct2 = {"b": 3, "c": 4}
dct3 = {**dct1, **dct2}
dct3
```

```
{'a': 1, 'b': 3, 'c': 4}
```

То же самое с помощью оператора распаковки. Таким образом можно соединять более двух словарей.

defaultdict() и Counter()

```
from collections import defaultdict
from collections import Counter

dct = defaultdict()
c = Counter()
```

defaultdict() и Counter() - удобные расширения на основе стандартных питоновских словарей, которые содержатся в модуле collections.

Для того, чтобы использовать эти расширения, необходимо импортировать их из модуля collections.

defaultdict(int)

```
from collections import defaultdict

dct = defaultdict(int)
dct["one"] += 1
dct["two"] += 2
dct
```

```
defaultdict(int, {'one': 1, 'two': 2})
```

defaultdict() принимает тип данных. После его создания можно производить операции со значениями несуществующих ключей, соответствующих этому типу данных – ошибки не будет. В случае с int дефолтным значением будет 0.

defaultdict(list)

```
from collections import defaultdict

dct = defaultdict(list)
dct["lst1"].append(1)
dct["lst2"].append(2)
dct
```

```
defaultdict(list, {'lst1': [1], 'lst2': [2]})
```

В случае с list, дефолтным значением будет пустой список.
В случае с другими коллекциями – пустая коллекция.
В случае со str – пустая строка.
В случае с bool – False.

Counter()

```
from collections import Counter

lst = ["a", "a", "b", "c",
       "c", "c"]
Counter(lst)
```

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

Counter - это словарь-счетчик. Чтобы посчитать частоты встретившихся в итерируемом объекте элементов, достаточно этот объект положить в аргументы Counter при его создании. Словари-счетчики умеют складываться друг с другом и вычитаться!

Counter() - пример

```
from string import punctuation
from collections import Counter

text = '''Словари в Python - неупорядоченные коллекции с доступом по ключу.
Словари в Python еще называют хеш-таблицами.'''
text = [word.strip(punctuation).lower() for word in text.split()]
text2 = '''Можно обратиться к элементу словаря в Python по ключу.
По ключу можно получить значение.'''
text2 = [word.strip(punctuation).lower() for word in text2.split()]
frequency = Counter(text) + Counter(text2)
for key, value in sorted(frequency.items(), key=lambda x: (-x[1], x[0])):
  print(key, value)
```

```
python 3
в 3
ключу 3
по 3
можно 2
```

Этот код подсчитывает одинаковые словоформы в двух текстах, а затем складывает результаты.

Спасибо за внимание!

admvereshchagina@gmail.com