Глава 9. Словари

Словарь (dictionary) похож на список, но имеет более широкие возможности. В списке позиция (или индекс) имеет целочисленное значение, в словаре индекс может быть (почти) любого типа.

Вы можете представить словарь как отображение между множеством индексов (тут они называются *ключами*) и множеством значений. Каждый ключ (key) отображается на значение. Связь между ключом и значением называется *парой ключ-значение* (key-value pair) или иногда *записью* (item).

В качестве примера, мы создадим словарь, который отображает английские и испанские слова, таким образом, ключи и значения являются строками.

Функция *dict* создает новый словарь без записей. Т.к. *dict* - имя встроенной функции, вы должны исключить его из имен переменных.

```
>>> eng2sp = dict()
>>> print eng2sp
{}
```

Фигурные кавычки означают *пустой словарь*. Для добавления записей в словарь, можно использовать квадратные скобки:

```
>>> eng2sp['one'] = 'uno'
```

Эта строка создает запись, которая отображает ключ 'one' на значение 'uno'. Если мы распечатаем словарь снова, то увидим пару ключ-значение, разделенную двоеточием:

```
>>> print eng2sp
{'one': 'uno'}
```

Этот выходной формат совпадает с входным форматом. Например, можно создать новый словарь с тремя записями:

```
>>> eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
```

Если вы выведете на экран содержимое переменной *eng2sp*, то удивитесь:

```
>>> print eng2sp
{'three': 'tres', 'two': 'dos', 'one': 'uno'}
```

Порядок пар ключ-значение не совпадает. Фактически, если вы наберете этот же пример на своем компьютере, то можете получить совершенно другой результат. Порядок записей в словаре непредсказуем.

Но это не проблема, т.к. элементы словаря никогда не индексируются числовыми индексами. Вместо этого вы используете ключи для поиска соответствующих значений:

```
>>> print eng2sp['two']
dos
```

Ключ 'two' всегда отображается на значение 'dos', поэтому порядок записей не имеет значения.

Если ключ отсутствует в словаре, вы получите исключение:

```
>>> print eng2sp['four'] Traceback
(most recent call last):
  File "<pyshell#8>", line 1, in
    <module> print eng2sp['four']
KeyError: 'four'
```

Функция *len* работает для словарей, она возвращает число пар ключ-значение:

```
>>> len(eng2sp)
3
```

Оператор *in* работает для словарей, он сообщает о похожих ключах в словаре.

```
>>> 'one' in eng2sp
True
>>> 'uno' in eng2sp
```

```
False
```

Чтобы убедиться, что какое-то значение встречается в словаре, вы можете использовать метод *values*, который возвращает значения в виде списка, а затем использовать оператор *in*:

```
>>> vals = eng2sp.values()
>>> 'uno' in vals
True
```

Оператор *in* использует различные алгоритмы для списков и словарей. Для списков он использует алгоритм линейного поиска. Время поиска пропорционально длине списка.

Для словарей Python использует алгоритм *хеш-таблиц* (hash table), который имеет замечательные свойства; *in* оператор занимает примерно столько же времени, сколько записей есть в словаре. Я не буду объяснять, почему хэш-функции, такие хорошие, но вы можете узнать об этом подробнее: wikipedia.org/wiki/Hash_table

9.1. Словарь как набор счетчиков

Предположим, вы хотите подсчитать, сколько раз каждая буква встречается в строке.

Есть несколько способов, чтобы это сделать:

1. Вы можете создать 26 переменных, по одной на каждую букву алфавита. Затем можно обойти строку и для каждого символа увеличивать соответствующий счетчик на единицу, возможно с использованием условных выражений (chained conditional).

2. Вы можете создать список из 26 элементов. Затем можно перевести каждый символ в число (с помощью встроенной функцию *ord*), используя число как индекс в списке и инкрементируя соответствующий счетчик.

3. Вы можете создать словарь с символами в виде ключей и счетчиками в виде значений. При первой встрече символа, вы должны добавить

запись в словарь. После этого можно инкрементировать значение существующей записи.

Каждый из этих вариантов выполняет одинаковые вычисления, но каждый из них реализует разные способы вычислений.

Реализация (implementation) - это способ выполнения вычислений, некоторые реализации лучше остальных. Например, преимущество реализации через словарь в том, что мы не должны знать заранее, какие буквы появятся в строке, у нас есть только свободное место для букв, которые появятся.

Код может выглядеть следующим образом:

```
def histogram(s):
    d = dict()
    for c in s:
        if c not in d:
            d[c] = 1
        else:
            d[c] = d[c] + 1
    return d
```

Имя функции - гистограмма, т.к. это статистический термин для множества счетчиков (или частот).

Первая строка функции создает пустой словарь. Цикл *for* обходит строку. Всякий раз в цикле, если символ *c* не встречается в словаре, мы создаем новую запись с ключом *c* и присваиваем начальное значение 1 (т.к. мы встретили эту букву один раз). Если *c* уже есть в словаре, то мы инкрементируем *d[c]*.

Так это работает:

```
>>> h = histogram('brontosaurus')
>>> print h
{'a': 1, 'b': 1, 'o': 2, 'n': 1, 's': 2, 'r': 2, 'u': 2, 't': 1}
```

9.2. Словари и файлы

Одно из распространенных применений словаря заключается в подсчете вхождений слов в текстовом файле. Давайте начнем с очень простого файла

слов, взятого из текста Ромео и Джульетты:
http://shakespeare.mit.edu/Tragedy/romeoandjuliet/romeo_juliet.2.2.html

В первом множестве примеров мы будем использовать сокращенную и упрощенную версию текста без знаков препинания. Позже мы будем работать с текстом сцены, включающим знаки препинания.

```
But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief
```

Мы напишем программу на Python, которая читает строки файла, разбивает каждую строку на список слов, затем проходит в цикле через каждое слово в строке и с использованием словаря подсчитывает слова.

Вы увидите, что получится два цикла *for*. Внешний цикл для строк файла и внутренний цикл для каждого слова в отдельной строке. Это пример шаблона, который называется *вложенные циклы* (nested loops), т.к. один из циклов *внешний* (outer), а второй - *внутренний* (inner).

Комбинация из двух вложенных циклов гарантирует, что мы будем считать каждое слово в каждой строке входного файла.

```
fname = raw_input('Enter the file name: ')
try:
    fhand = open(fname)
except:
    print 'File cannot be opened:',
    fname exit()

counts = dict()
for line in fhand:
    words = line.split()
    for word in words:
        if word not in counts:
           counts[word] = 1
        else:
           counts[word] += 1
print counts
```

Когда мы запускаем программу, мы видим сырой дамп (raw dump) всех счетчиков в неотсортированном хэш порядке (файл romeo.txt file доступен по адресу: pycode.ru/files/python/romeo.txt))

```
Enter the file name: romeo.txt
{'and': 3, 'envious': 1, 'already': 1, 'fair': 1, 'is': 3,
'through': 1, 'pale': 1,

'yonder': 1, 'what': 1, 'sun': 2, 'Who': 1, 'But': 1, 'moon': 1,
'window': 1,

'sick': 1, 'east': 1, 'breaks': 1, 'grief': 1, 'with': 1,
'light': 1, 'It': 1,

'Arise': 1, 'kill': 1, 'the': 3, 'soft': 1, 'Juliet': 1}
```

Немного неудобно просматривать словарь в поисках наиболее употребительных слов и их количества, поэтому нам нужно добавить еще несколько строк кода, чтобы получить более полезный результат на выходе.

9.3. Циклы и словари

Если вы используете словарь как *последовательность* (sequence) в операторе *for*, происходит обход ключей словаря. Например, *print_hist* выводит на экран каждый ключ и соответствующее ему значение:

```
def print_hist(h):
    for c in h:
        print c, h[c]
```

Результат имеет следующий вид:

```
>>> h = histogram('parrot')
>>> print_hist h
a 1
p 1
r 2
t 1
o 1
```

Снова ключи не имеют порядка.

Если вы хотите вывести на экран ключи в алфавитном порядке, сначала составьте список ключей в словаре с помощью метода словаря *keys*, затем отсортируйте этот список и в цикле, просматривая каждый ключ, выводите на экран пару ключ/значение, как показано далее:

```
def print_sorted_hist(h):
    lst = h.keys()
    lst.sort()
    for c in lst:
        print c, h[c]
```

Результат будет следующий:

```
>>> h = histogram('parrot')
>>> print_sorted_hist(h)
a 1
o 1
p 1
r 2
t 1
```

Теперь ключи отсортированы в алфавитном порядке.