Словари

Начнем с рассмотрения примера - *В файле находится список слов, среди которых есть повторяющиеся. Каждое слово записано в отдельной строке. Построить алфавитно-частотный словарь: все различные слова должны быть записаны в другой файл в алфавитном порядке, справа от каждого слова указано, сколько раз оно встречается в исходном файле.*

Для решения задачи нам нужно составить особую структуру данных – *словарь* (англ. *dictionary*), в котором хранить пары «слово – количество». Таким образом, мы хотим искать нужные нам данные не по числовому индексу элемента (как в списке), а по слову (символьной строке). Например, вывести на экран количество найденных слов «бегемот» можно было бы так:

```
print ( D["бегемот"] )
```

где **D** – имя словаря.

Типы данных для построения словаря есть в некоторых современных языках программирования. В Python для работы со словарями есть встроенный тип данных **dict** (от англ. *dictionary*).

Словарь – это неупорядоченный набор элементов, в котором доступ к элементу выполняется по ключу.

Слово «неупорядоченный» в этом определении говорит о том, что порядок элементов в словаре никак не задан, он определяется внутренними механизмами хранения данных языка. Поэтому сортировку словаря выполнить невозможно, как невозможно указать для какого-то элемента его соседей (предыдущий и следующий элементы).

Ключом может быть любой неизменяемый тип данных, например, число, символьная строка или *кортеж* (неизменяемый набор значений). В одном словаре можно использовать ключи разных типов.

Алфавитно-частотный словарь

Вернемся к нашей задаче построения алфавитно-частотного словаря. Алгоритм, записанный в виде псевдокода, может выглядеть так:

```
создать пустой словарь
while есть слова в файле:
  прочитать очередное слово
  if слово есть в словаре:
    увеличить на 1 счётчик для этого слова
  else:
    добавить слово в словарь
    записать 1 в счетчик слова
```

Теперь нужно записать все шаги этого алгоритма с помощью операторов языка программирования.

Словарь определяется с помощью фигурных скобок, например,

```
D = { "бегемот": 0, "пароход": 2 }
```

В этом словаре два элемента, ключ «бегемот» связан со значением 0, а ключ «пароход» – со значением 2. Пустые фигурные скобки задают пустой словарь:

```
D = {}
```

Для того, чтобы добавить элемент в словарь, используют присваивание:

```
D["самолёт"] = 1
```

Если ключ «самолёт» уже есть в словаре, соответствующее значение будет изменено, а если такого ключа нет, то он будет добавлен и связан со значением 1. Нам нужно увеличивать значение счётчика слов на 1, это можно сделать так:

```
D["самолёт"] += 1
```

Однако, если ключа «самолёт» нет в словаре, такой оператор вызовет ошибку. Для того, чтобы определить, есть ли в словаре какой-то ключ, можно использовать оператор **in**:

```
if "самолёт" in D:
  D["самолёт"] += 1
else:
  D["самолёт"] = 1
```

Если есть ключ «самолёт», соответствующее значение увеличивается на 1. Если такого ключа нет, то он создается и связывается со значением, равным 1.Можно обойтись вообще без условного оператора, если использовать метод **get** для словаря, который возвращает значение, связанное с существующим ключом. Если ключа нет в словаре, метод возвращает значение по умолчанию, которое задаётся как второй параметр:

```
D["самолёт"] = D.get ( "самолёт", 0 ) + 1
```

В данном случае значение по умолчанию – 0, если ключа «самолёт» нет, создаётся элемент с таким ключом и соответствующее значение будет равно 1. Теперь у нас есть всё для того, чтобы написать полный цикл ввода данных и составления списка:

```
D = {}
F = open ( "input.txt" )
while True:
  word = F.readline().strip()       # (*)
  if not word: break
  D[word] = D.get ( word, 0 ) + 1
F.close()
```

Обратим внимание на строку (*) в программе. После чтения очередной строки из файла **F** вызывается еще и метод **strip** (от англ. лишать, удалять), который удаляет лишние пробелы и завершающий символ перевода строки «**\n**».

Теперь остаётся вывести результат в файл. В отличие от списка, к элементам словаря нельзя обращаться по индексам. Тогда возникает вопрос – как же перебрать все возможные ключи? Для этой цели мы запросим у словаря список всех ключей, используя метод **keys**:

```
allKeys = D.keys()
```

Эти ключи нужно отсортировать, тут работает функция **sorted**, которая вернёт отсортированный список:

```
sortKeys = sorted(D.keys())
```

В последних версиях языка Python вместо этого можно записать просто

```
sortKeys = sorted(D)
```

не вызывая явно метод **keys**.

Остаётся только перебрать в цикле **for** все элементы этого списка, например, так:

```
F = open ( "output.txt", "w" )
for k in sorted(D):
  F.write ( "{}: {}\n".format(k, D[k]) )
F.close()
```

Ключи из отсортированного списка ключей попадают по очереди в переменную k, для каждого ключа выводится сам ключ и через двоеточие – связанное с ним значение, то есть количество таких слов в исходном файле. Для того, чтобы преобразовать данные перед выводом в символьную строку, используется функция **format**. Две пары фигурных скобок в строке форматирования обозначают места для вывода первого и второго аргументов функции.

Отметим, что у словарей есть метод **values**, который возвращает список значений в словаре. Например, вот так можно вывести значения для всех ключей:

```
for i in D.values():
  print ( i )
```

Если же нас интересуют пары «ключ-значение», удобно использовать метод **items**, который возвращает список таких пар. Перебрать все пары и вывести их на экран можно с помощью следующего цикла:

```
for k, v in D.items():
  print ( k, "->", v )
```

В этом цикле две изменяемых переменных: **k** (ключ) и **v** (значение). Поскольку **D.items()** – это список пар «ключ-значение», при переборе первый элемент пары (ключ) попадает в переменную **k**, а второй (значение) – в переменную **v**.