Dictionaries

Chapter 11

A Dictionary is like a List

- Like a `list`, but more general. Indices can be (almost) any type
- Contains a collection of indices (**keys**) and their corresponding **values**.

List

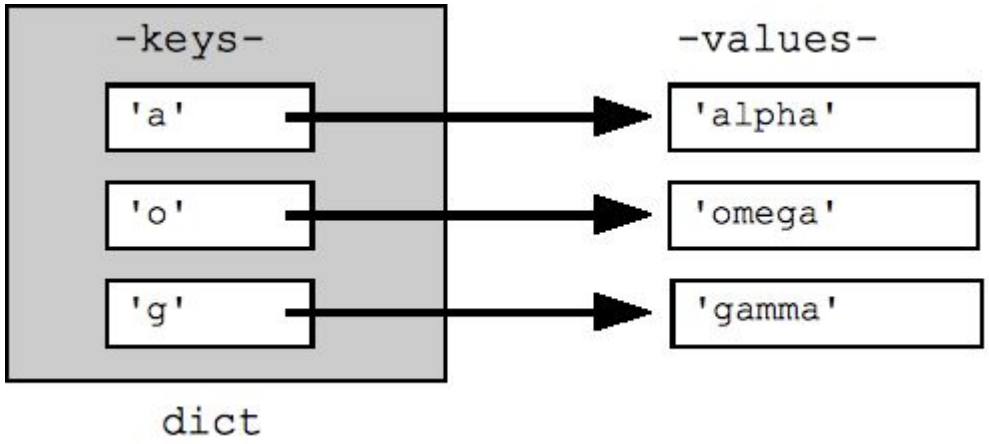
index	value
0	"Eggs"
1	"Milk"
2	"Cheese"
3	"Yogurt"
4	"Butter"
5	"More Cheese"

Dictionary

key	value
'Eggs'	2.59
'Milk'	3.19
'Cheese'	4.80
'Yogurt'	1.35
'Butter'	2.59
'More Cheese'	6.19

A Dictionary is a Mapping

- Each key is associated with a single value. This association is called a **key-value pair** or an **item**.
- Each key “maps to” a value.



Creating a Dictionary

```
>>> eng2sp = dict()      # creates a new dictionary

>>> eng2sp
{}                       # empty dictionary

>>> eng2sp[' one' ] = ' uno'   # entering an item

>>> eng2sp
{' one' : ' uno' }       # key:value pair
```

Creating a Dictionary (cont.)

```
>>> eng2sp = {'one' : 'uno', 'two':'dos', 'three':'tres'}
# creating a dictionary with items

>>> eng2sp
{' one' : ' uno' , ' three' : ' tres' , ' two' : ' dos' }
# what about the order?

>>> eng2sp[' two' ]
' dos'

>>> eng2sp['four' ]
KeyError: 'four'
```

Creating a Dictionary (cont.2)

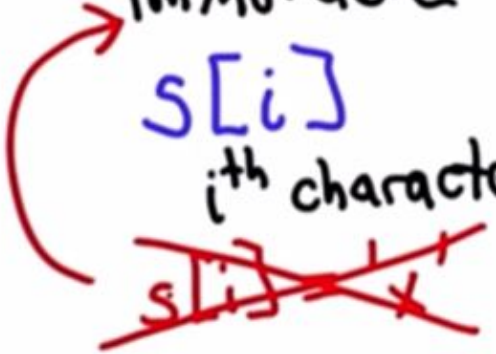
```
>>> len(eng2sp)      # number of items (key-value pairs)
3

>>> ' one' in eng2sp        # search for a key
True
>>> ' uno' in eng2sp
False

>>> vals = eng2sp.values() # get all the values
>>> ' uno' in vals          # search for a value
True
```

String

'hello'

sequence of characters

immutable



s[i]

i^{th} character in s

s[i] = 'x'

List

['alpha', 23]

list of elements

mutable

p[i]

i^{th} element of p

p[i] = v

replace value of i^{th} element with v

Dictionary

{'hydrogen': 1, 'helium': 23}

set of <key, value> pairs

mutable

d[k]

value associated with k in d

key

d[k] = v

update k → v

Dictionary as a Collection of Counters

Problem:
Given a string, count how many times each letter appears.

Ex:

```
>>> count('banana')
b : 1
a : 3
n : 2
```

Dictionary as a Collection of Counters (cont.)

Solution-1:
Create 26 variables. Traverse the string, increment the corresponding counter. Use chained conditionals.

Solution-2:
Create a list of 26 elements. Convert each character to a number. Use the number as index, and increment the appropriate counter.

Solution-3:
Create a dictionary with characters as keys and counters as the corresponding values. The first time you see a character, add to list. Next time, just increment.

Dictionary as a Collection of Counters (cont.2)

- Each of the above solutions performs the same computation, but each of them implements that computation in a different way.

- An **implementation** is a way of performing a computation; some implementations are better than others.

Dictionary as a Collection of Counters (cont.3)

```
def histogram(s):    # s is a string
    d = dict()
    for c in s:      # c is a character
        if c not in d:   # add new if doesn't exist
            d[c] = 1
        else:            # otherwise, increment the existing one
            d[c] += 1
    return d
```

Dictionary as a Collection of Counters (cont.4)

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

get method

```
>>> h = histogram('a')
>>> h
{'a' : 1}

>>> h.get('a' , 0)   # 0 is the default value
1

>>> h.get('b' , 0)
0
```

Looping and Dictionaries

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

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

Looping and Dictionaries (cont.)

- The above output has no particular order.

```
>>> for key in sorted(h):
... print(key, h[key])

a 1
o 1
p 1
r 2
t 1
```