dictionaries
(aka hash tables or hash maps)

Genome 559: Introduction to Statistical and Computational Genomics

Prof. James H. Thomas

Review

- You should be comfortable with loops by now. I will posted more problems on loops if you need practice.

- Pay attention to program robustness and speed.

- Account for very large or very small input files (or lists, etc).

- Consider handling files or command-line options with the wrong format by giving useful feedback.

Dictionaries

- A dictionary organizes linked information

- Examples:
  - word and definition
  - name and phone number
  - name and DNA sequence
  - username and password

- If you know the first entry, you can quickly and easily get the second one

Rules for dictionaries

- The first item is a "key"
- Each key can appear only once
- A key must be an <u>immutable</u> object: number, string, or tuple
- Lists cannot be keys (they are mutable)
- The key is the item you'll use to do look-ups

Key examples

Phone book: we have a name, we want a number

Name is the key

Crank call prevention: we have a number, we want a name

Number is the key

Creating a dictionary

```
#create an empty dictionary
myDict = {}

#create a dictionary with three entries
myDict = {"Curly":4123, "Larry":2057, "Moe":1122}

#add another entry
myDict["Shemp"] = 2232

#change Moe's phone number
myDict["Moe"] = 4040

#delete Moe from dictionary
del myDict["Moe"]
```

Using a dictionary

```
>>> myDict = {"Curly":4123, "Larry":2057, "Moe":1122}
>>> myDict["Moe"]
1122
>>> myDict.keys()
['Larry', 'Moe', 'Curly']
>>> "Curly" in myDict
True
>>> "curly" in myDict
False
>>> myDict.values()
[2057, 1122, 4123]
>>> len(myDict)
3
```

unlike a list, the key:value pairs are not in any particular order

curly is not the same as Curly

Using a dictionary

```
birthdays = { "George":"June 12", "W":"July 6", "Barack":"Aug 4" }
for person in birthdays.keys():
    print "Send", person, "a card on", birthdays[person]
```

or possibly

```
for person in birthdays.keys():
    if person == "Barack":
        print "Send", person, "a card on", birthdays[person]
    else:
        print "Send", person, "an insult on", birthdays[person]
```

`dictionary.keys()` returns a <u>list</u> of the keys!

Sorting a dictionary

```
sortedkeys = birthday.keys()
sortedkeys.sort()
for person in sortedkeys:
    print "Send", person, "a card on", birthdays[person]
```

Use the list.sort() method - if the list contains strings, they will be sorted alphanumerically

Making a useful dictionary

Suppose we have a file that gives the alignment score for a large number of sequences:

seq1 <tab> 37
seq2 <tab> 182
etc.

```
import sys
myFile = open(sys.argv[1], "r")
scoreDict = {}
for line in myFile:
    fields = line.strip().split("\t")
    scoreDict[fields[0]] = float(fields[1])
myFile.close()
```

we now have a dictionary where we can look up a score for any name

dict basics

```
D = {'dna':'T','rna':'U'}   # dictionary literal assignment
D = {}                      # make an empty dictionary
D.keys()                    # get the keys as a list
D.values()                  # get the values as a list
D['dna']                    # get a value based on key
D['dna'] = 'T'              # set a key:value pair
del D['dna']                # delete a key:value pair
D.pop('dna')                # remove key:value (and return value)
'dna' in D                  # True if key 'dna' is found in D, else False
```

The keys must be immutable objects (e.g. string, int, tuple).

The values can be anything (including a list or another dictionary).

The order of elements in the list returned by **`D.keys()`** or **`D.values()`** is arbitrary (effectively random).

Each key can be stored only once in the dictionary, so if you set the value for a key for a second time it OVERWRITES the old value!

Sample problem #1

The file "scores.txt" (linked from news on web site) contains blastn scores for a large number of sequences with a particular query. Write a program that reads them into a dictionary, <u>sorts them by sequence name</u>, and prints them.

```
>python sort_dict.py scores.txt
seq00000      293
seq00001      315
seq00002      556
seq00003      556
seq00004      617
seq00005      158
etc.
```

Solution #1

```
import sys
myFile = open(sys.argv[1], "r")

# make an empty dictionary and populate it
scoreDict = {}
for line in myFile:
    fields = line.strip().split("\t")
    # record each value with name as key
    scoreDict[fields[0]] = float(fields[1])
myFile.close()

# get key list and sort it
keys = scoreDict.keys()
keys.sort()

# print based on sorted keys
for key in keys:
    print key + "\t" + scoreDict[key]
```

Sample problem #2

Suppose you have a list of sequence names whose scores you are interested in extracting from the large list of scores (in the same file "scores.txt"). Modify your previous program to read the list of sequence names from a second file and print just those values. A sample "seq_names.txt" is linked from the web site.

```
>python get_scores.py scores.txt seq_names.txt
seq00036     784
seq57157     523
seq58039     517
seq67160     641
seq76732     44
seq83199     440
seq92309     446
```

Solution #2

```
import sys

# get a list of the names of interest
seqNameFile = open(sys.argv[2], "r")
seqNameList = []
for line in seqNameFile:
    seqNameList.append(line.strip())
seqNameFile.close()

# make a dictionary of the scores, keyed on name
dictFile = open(sys.argv[1], "r")
scoreDict = {}
for line in dictFile:
    fields = line.strip().split("\t")
    scoreDict[fields[0]] = int(fields[1])
dictFile.close()

# finally, use the dictionary
for seqName in seqNameList:
       print seqName + "\t" +  str(scoreDict[seqName])
```

Challenge problems

1. Extend your program in sample problem 2 so that it gives useful user feedback when a sequence name is missing from the dictionary.

2. Sort the list of scores in the same file (scores.txt) by <u>score</u>, with the highest scoring first. Print the sequence name and its score in that order. You can do this using a dictionary (ignore the fact that more than one sequence may have the same score, so some may get lost).

Challenge 1 solution

```
import sys
# get a list of the names of interest
seqNameFile = open(sys.argv[2], "r")
seqNameList = []
for line in seqNameFile:
    seqNameList.append(line.strip())
seqNameFile.close()
# make a dictionary of the scores, keyed on name
dictFile = open(sys.argv[1], "r")
scoreDict = {}
for line in dictFile:
    fields = line.strip().split("\t")
    scoreDict[fields[0]] = int(fields[1])
dictFile.close()
# finally, use the dictionary
for seqName in seqNameList:
    if not seqName in scoreDict:
        print seqName, "not found"
    else:
        print seqName + "\t" +  scoreDict[seqName]
```

Challenge 2 solution

```
import sys
dictFile = open(sys.argv[1], "r")

scoreDict = {}
for line in dictFile:
    fields = line.strip().split("\t")
    scoreDict[int(fields[1])] = fields[0]
dictFile.close()

sortKeys = scoreDict.keys()
sortKeys.sort()
sortKeys.reverse() # sort makes ascending sort for numbers

for key in sortKeys:
    print scoreDict[key] + "\t" + key
```

How a hash map (hash table) works (FYI)

- the hash map (dictionary in Python) is a great example of a clever data structure. Refer to the figure on next slide.

- under the hood, the hash map is simply a list, with each value stored at a specific position in the list. The list is typically quite a bit longer than the number of values to be stored (the extra list elements are empty).

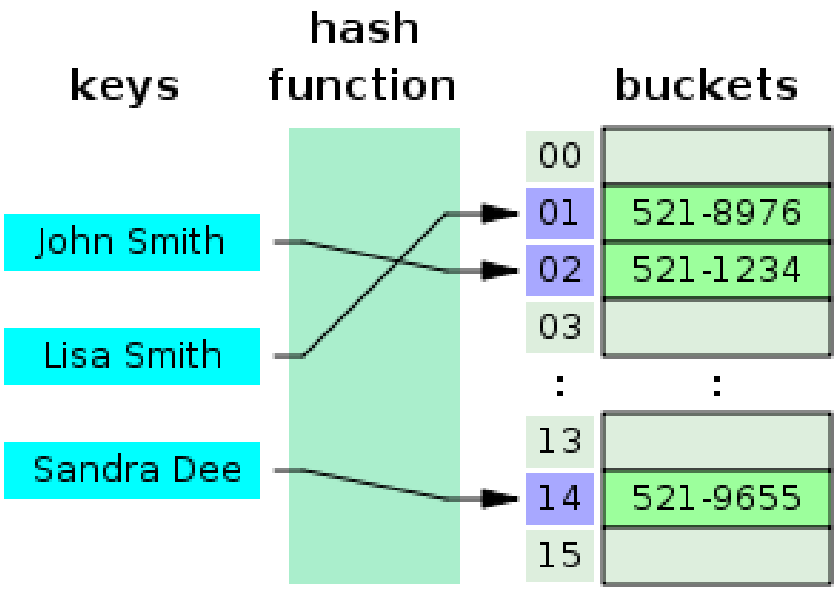
- when a key is presented to the hash map it is converted to an integer that corresponds to where its value is stored in the list.

- the value at that list position is stored or returned (recall that getting an element at a specific list position is very fast because of the way computer memory is arranged).

- the algorithm by which the key is converted to an integer is called a "hash function", hence the name hash map.

- the speed of accessing a hash map is dominated by how fast and "well-distributed" the hash function is (they are very fast).

- this is a slight simplification, in particular it doesn't account for hash collisions (see Wikipedia article).



(list elements in a hash map are traditionally called buckets)

this hash map has 3 of 16 buckets filled, the rest are empty