

Python Dictionaries

Chapter 9



Python for Informatics: Exploring Information
www.pythonlearn.com

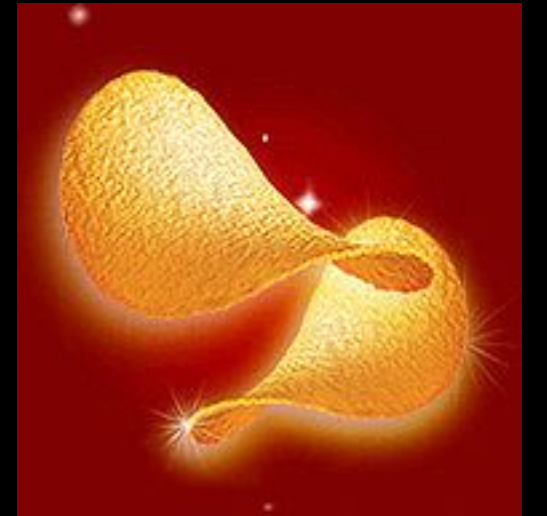




A Story of Two Collections..

- List

- › A linear collection of values that stay in order



- Dictionary

- › A “bag” of values, each with its own label



Dictionaries



- Dictionaries are Python's most powerful data collection
- Dictionaries allow us to do fast database-like operations in Python
- Dictionaries have different names in different languages
 - › Associative Arrays – Perl / PHP
 - › Properties or Map or HashMap – Java
 - › Property Bag – C# / .Net

http://en.wikipedia.org/wiki/Associative_array

Dictionaries

- Lists **index** their entries based on the position in the list
- **Dictionaries** are like bags - no order
- So we **index** the things we put in the **dictionary** with a “lookup tag”

```
>>> purse = dict()
>>> purse['money'] = 12
>>> purse['candy'] = 3
>>> purse['tissues'] = 75
>>> print purse
{'money': 12, 'tissues': 75, 'candy': 3}
>>> print purse['candy']
3
>>> purse['candy'] = purse['candy'] + 2
>>> print purse
{'money': 12, 'tissues': 75, 'candy': 5}
```

Comparing Lists and Dictionaries

- Dictionaries are like lists except that they use keys instead of numbers to look up values

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print lst
[21, 183]
>>> lst[0] = 23
>>> print lst
[23, 183]
```

```
>>> ddd = dict()
>>> ddd['age'] = 21
>>> ddd['course'] = 182
>>> print ddd
{'course': 182, 'age': 21}
>>> ddd['age'] = 23
>>> print ddd
{'course': 182, 'age': 23}
```

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print lst
[21, 183]
>>> lst[0] = 23
>>> print lst
[23, 183]
```

```
>>> ddd = dict()
>>> ddd['age'] = 21
>>> ddd['course'] = 182
>>> print ddd
{'course': 182, 'age': 21}
>>> ddd['age'] = 23
>>> print ddd
{'course': 182, 'age': 23}
```

List

Key	Value
-----	-------

[0]	21
[1]	183

lst

Dictionary

Key	Value
-----	-------

['course']	182
['age']	21

ddd

Dictionary Literals (Constants)

- Dictionary literals use curly braces and have a list of **key** : **value** pairs
- You can make an **empty dictionary** using empty curly braces

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print jjj
{'jan': 100, 'chuck': 1, 'fred': 42}
>>> ooo = { }
>>> print ooo
{}
>>>
```

Most Common Name?

marquard	cwen	cwen
zhen	marquard	zhen
csev	zhen	csev
zhen	csev	marquard
		zhen

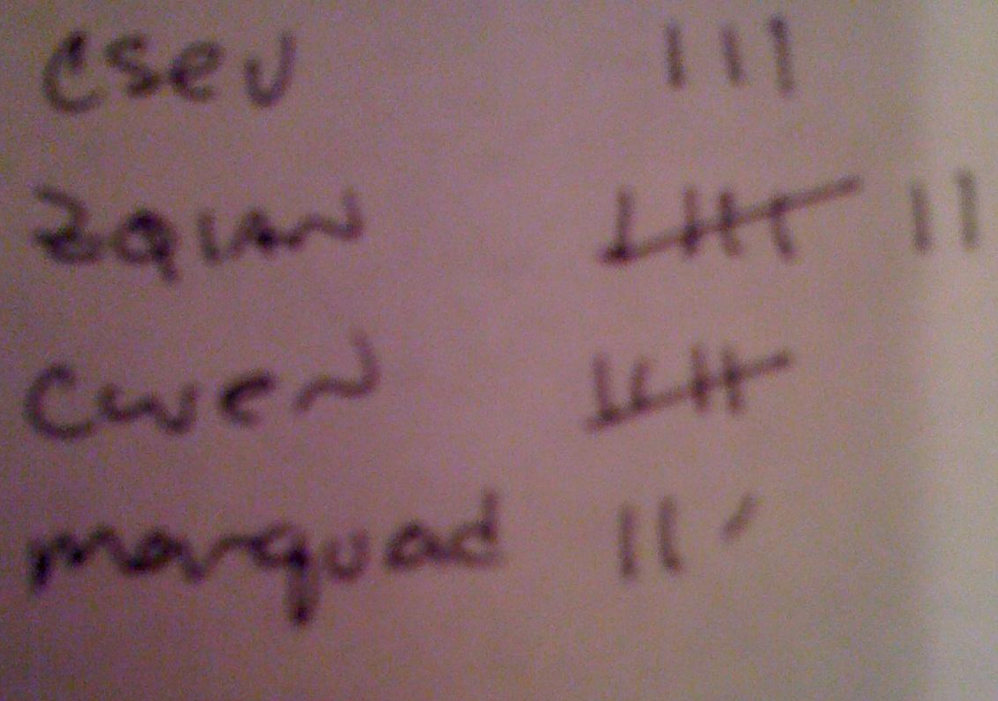
Many Counters with a Dictionary

- One common use of dictionary is **counting** how often we “see” something

```
>>> ccc = dict()
>>> ccc['csev'] = 1
>>> ccc['cwen'] = 1
>>> print ccc
{'csev': 1, 'cwen': 1}
>>> ccc['cwen'] = ccc['cwen'] + 1
>>> print ccc
{'csev': 1, 'cwen': 2}
```

Key

Value



csev	
zqian	
cwen	
marquard	

Dictionary Tracebacks

- It is an **error** to reference a key which is not in the dictionary
- We can use the **in** operator to see if a key is in the dictionary

```
>>> ccc = dict()
>>> print ccc['csev']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'csev'
>>> print 'csev' in ccc
False
```

When we see a new name

- When we encounter a new name, we need to add a new entry in the **dictionary** and if this the second or later time we have seen the **name**, we simply add one to the count in the **dictionary** under that **name**

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names :
    if name not in counts:
        counts[name] = 1
    else :
        counts[name] = counts[name] + 1
print counts
```

{'csev': 2, 'zqian': 1, 'cwen': 2}

The `get` method for dictionaries

- This pattern of checking to see if a `key` is already in a dictionary and assuming a default value if the `key` is not there is so common, that there is a `method` called `get()` that does this for us

```
if name in counts:  
    x = counts[name]  
else :  
    x = 0
```

```
x = counts.get(name, 0)
```

Default value if key does not exist
(and no Traceback).

```
{'csev': 2, 'zqian': 1, 'cwen': 2}
```

Simplified counting with `get()`

- We can use `get()` and provide a default value of zero when the key is not yet in the dictionary - and then just add one

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names :
    counts[name] = counts.get(name, 0) + 1
print counts
```

Default



`{'csev': 2, 'zqian': 1, 'cwen': 2}`

Simplified counting with `get()`

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names:
    counts[name] = counts.get(name, 0) + 1
print counts
```



<http://www.youtube.com/watch?v=EHJ9uYx5L58>

Writing programs (or programming) is a very creative and rewarding activity. You can write programs for many reasons ranging from making your living to solving a difficult data analysis problem to having fun to helping someone else solve a problem. This book assumes that everyone needs to know how to program and that once you know how to program, you will figure out what you want to do with your newfound skills.

We are surrounded in our daily lives with computers ranging from laptops to cell phones. We can think of these computers as our "personal assistants" who can take care of many things on our behalf. The hardware in our current-day computers is essentially built to continuously ask us the question, "What would you like me to do next?"

Our computers are fast and have vast amounts of memory and could be very helpful to us if we only knew the language to speak to explain to the computer what we would like it to "do next". If we knew this language we could tell the computer to do tasks on our behalf that were repetitive. Interestingly, the kinds of things computers can do best are often the kinds of things that we humans find boring and mind-numbing.

Counting Pattern

```
counts = dict()
print 'Enter a line of text:'
line = raw_input('')

words = line.split()

print 'Words:', words

print 'Counting...'
for word in words:
    counts[word] = counts.get(word,0) + 1
print 'Counts', counts
```

The general pattern to count the words in a line of text is to **split** the line into words, then loop through the words and use a **dictionary** to track the count of each word independently.

Counting Words



```
python wordcount.py
```

```
Enter a line of text:
```

```
the clown ran after the car and the car ran into the tent  
and the tent fell down on the clown and the car
```

```
Words: ['the', 'clown', 'ran', 'after', 'the', 'car',  
'and', 'the', 'car', 'ran', 'into', 'the', 'tent', 'and',  
'the', 'tent', 'fell', 'down', 'on', 'the', 'clown',  
'and', 'the', 'car']
```

```
Counting...
```

```
Counts {'and': 3, 'on': 1, 'ran': 2, 'car': 3, 'into': 1,  
'after': 1, 'clown': 2, 'down': 1, 'fell': 1, 'the': 7,  
'tent': 2}
```

<http://www.flickr.com/photos/71502646@N00/2526007974/>

```
counts = dict()
print 'Enter a line of text:'
line = raw_input('')
words = line.split()

print 'Words:', words
print 'Counting...'

for word in words:
    counts[word] = counts.get(word,0) + 1
print 'Counts', counts
```



python wordcount.py

Enter a line of text:

the clown ran after the car and the car ran
into the tent and the tent fell down on
the clown and the car

Words: ['the', 'clown', 'ran', 'after', 'the',
'car', 'and', 'the', 'car', 'ran', 'into', 'the',
'tent', 'and', 'the', 'tent', 'fell', 'down', 'on',
'the', 'clown', 'and', 'the', 'car']

Counting...

Counts {'and': 3, 'on': 1, 'ran': 2, 'car': 3,
'into': 1, 'after': 1, 'clown': 2, 'down': 1, 'fell':
1, 'the': 7, 'tent': 2}

Definite Loops and Dictionaries

- Even though **dictionaries** are not stored in order, we can write a **for** loop that goes through all the **entries** in a **dictionary** - actually it goes through all of the **keys** in the **dictionary** and **looks up** the values

```
>>> counts = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> for key in counts:
...     print key, counts[key]
...
jan 100
chuck 1
fred 42
>>>
```

Retrieving lists of Keys and Values

- You can get a list of **keys**, **values**, or **items (both)** from a dictionary

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print list(jjj)
['jan', 'chuck', 'fred']
>>> print jjj.keys()
['jan', 'chuck', 'fred']
>>> print jjj.values()
[100, 1, 42]
>>> print jjj.items()
[('jan', 100), ('chuck', 1), ('fred', 42)]
>>>
```



What is a 'tuple'? - coming soon...

Bonus: Two Iteration Variables!

- We loop through the **key-value** pairs in a dictionary using **two** iteration variables
- Each iteration, the first variable is the **key** and the second variable is the *corresponding value* for the key

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42,
            'jan': 100}
>>> for aaa,bbb in jjj.items() :
...     print aaa, bbb
...
jan 100
chuck 1
fred 42
>>>
```

aaa	bbb
[jan]	100
[chuck]	1
[fred]	42

```
name = raw_input('Enter file:')
handle = open(name)
text = handle.read()
words = text.split()

counts = dict()
for word in words:
    counts[word] = counts.get(word,0) + 1

bigcount = None
bigword = None
for word,count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print bigword, bigcount
```

python words.py
Enter file: words.txt
to 16

python words.py
Enter file: clown.txt
the 7

Summary

- What is a collection?
- Lists versus Dictionaries
- Dictionary constants
- The most common word
- Using the `get()` method
- Hashing, and lack of order
- Writing dictionary loops
- Sneak peek: tuples
- Sorting dictionaries



Acknowledgements / Contributions



These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and open.umich.edu and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

... Insert new Contributors or translation credits here

...