

파이썬 딕셔너리

제9장

Python for Everybody
www.py4e.com



“컬렉션”이란?



- 컬렉션은 한 가지 이상의 값을 넣고 한꺼번에 가지고 돌아다닐 수 있어서 편리함
- 여러 개의 값을 하나의 “변수”에 담을 수 있음
- 변수 “안”에 공간을 여러 개 가짐
- 변수 안에서 서로 다른 공간을 찾는 방법이 있음

“컬렉션”이 아닌 것은?

대부분의 변수는 한 값을 갖음 - 변수에 새 값을 대입하면
- 이전 값에 덮어쓰워 짐

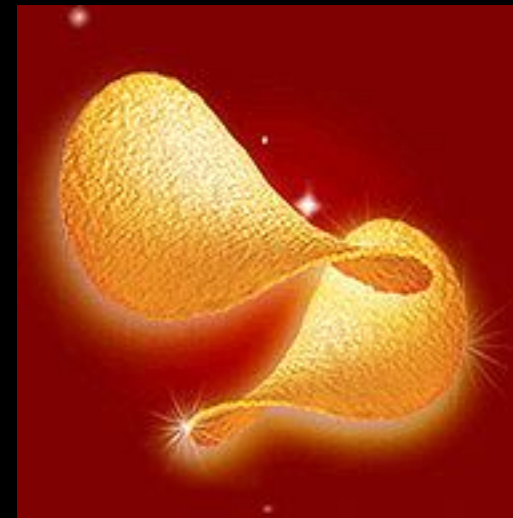
```
$ python
>>> x = 2
>>> x = 4
>>> print(x)
4
```



두 가지 컬렉션에 대해..

- 리스트

- 순서를 유지하는 값들의 선형 컬렉션



- 딕셔너리

- 고유의 라벨을 갖고 있는 값을 넣는 “가방”



딕셔너리



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

딕셔너리



- 파이썬의 가장 강력한 데이터 컬렉션
- 파이썬에서 빠르게 데이터베이스 같은 연산을 가능하게 함
- 다른 언어에서는 다른 이름으로 불림
 - Associative Arrays (연관 배열) - Perl / PHP
 - Properties or Map or HashMap (속성, 맵, 해쉬맵) - Java
 - Property Bag (속성 가방) - C# / .Net

딕셔너리

- 리스트는 리스트 안에서 원소의 위치를 기반으로 인덱스를 매김
- 딕셔너리는 가방과 같음 - 순서가 없음
- 따라서 딕셔너리에 넣는 대상은 “조화 태그”를 달아 인덱스를 매김

```
>>> 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}
```

리스트와 딕셔너리 비교

딕셔너리는 값을 찾기 위해 숫자 대신 키를 사용하는 것만 빼면
리스트와 동일

```
>>> 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]
```

리스트

키 값

[0]	21
[1]	183

lst

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

딕셔너리

키 값

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

ddd

딕셔너리 표현 (상수)

- 딕셔너리는 중괄호로 표현하며 **키** : **값** 쌍 목록을 가짐
- 사이가 비어있는 중괄호로 **빈 딕셔너리**를 만들 수 있음

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

가장 많이 나온 이름은?

가장 많이 나온 이름은?

marquard

cwen

cwen

zhen

marquard

zhen

csev

zhen

csev

marquard

zhen

csev

zhen

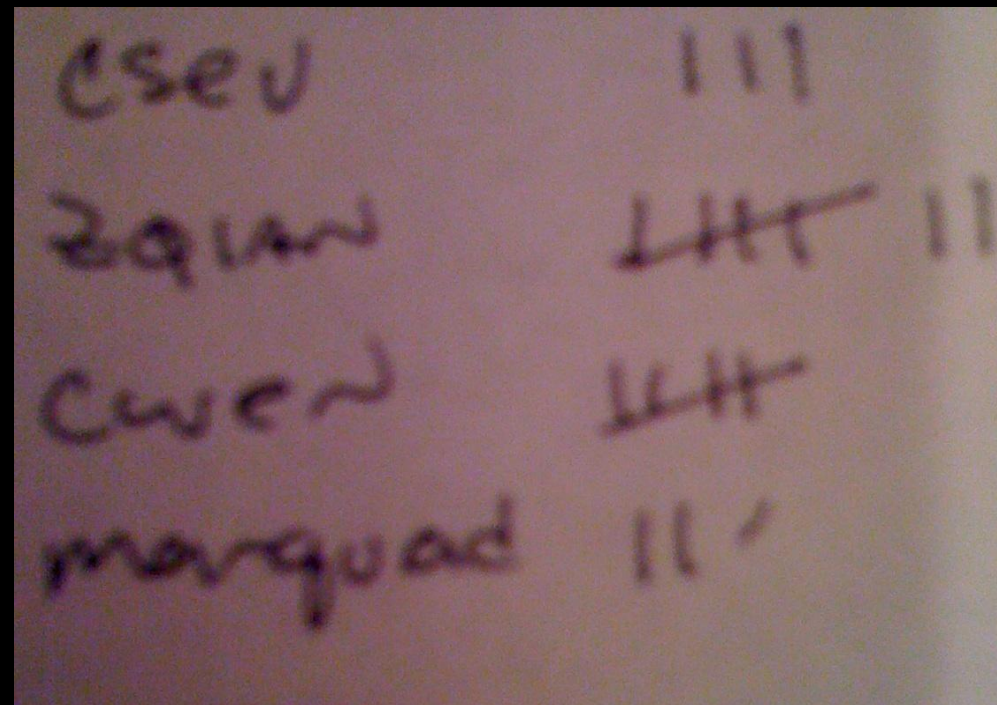
가장 많이 나온 이름은?

marquard

cwen

cwen

zhen



A photograph of a piece of paper with handwritten text. The text is organized into two columns. The left column lists names: 'csev', 'zhen', 'cwen', and 'marquard'. The right column lists corresponding counts: '111', '111', '111', and '111'. There are some additional marks and lines next to the counts, possibly indicating a total or a correction.

csev	111
zhen	111
cwen	111
marquard	111

zhen

csev

csev

zhen

csev

marquard

zhen

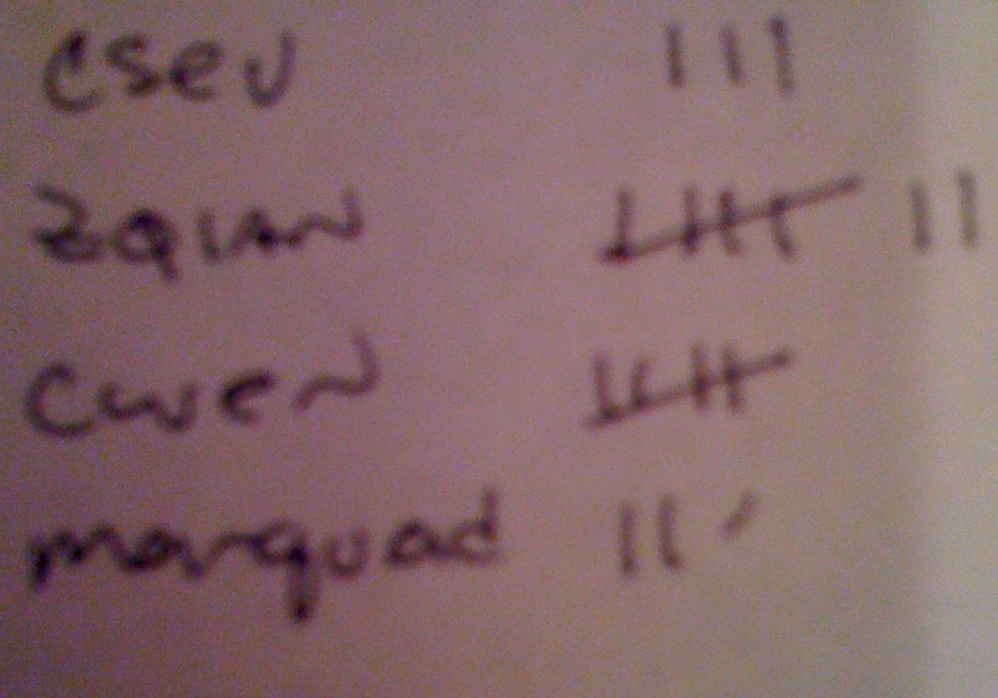
딕셔너리를 이용한 여러 카운터

딕셔너리의 일반적인 활용법 중 하나
: 대상이 얼마나 자주 보이는지를 “세는”
카운팅

```
>>> 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}
```

키

값



csev	111
zqian	11111111
cwen	1111
marquard	111

딕셔너리 Traceback 에러

- 딕셔너리에 없는 키를 참조하는 것은 오류를 일으킴
- `in` 연산자를 사용하여 키가 딕셔너리에 있는지 확인 가능

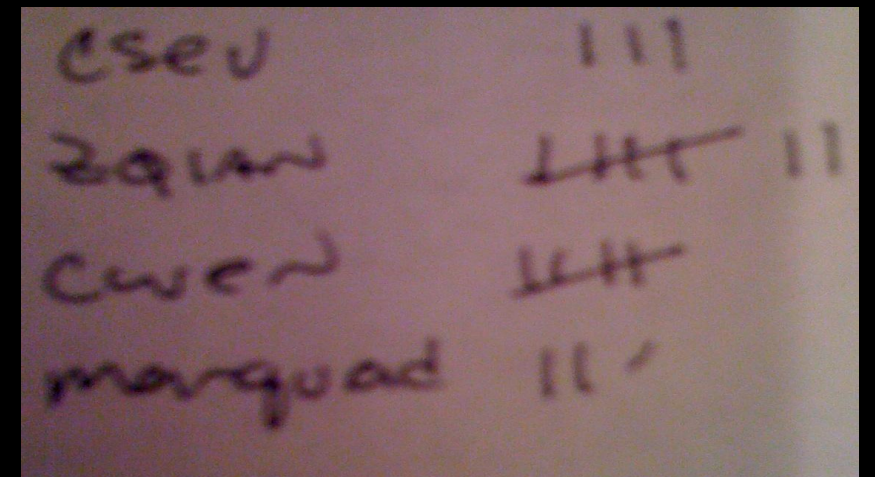
```
>>> ccc = dict()
>>> print(ccc['csev'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'csev'
>>> 'csev' in ccc
False
```


새로운 이름을 보는 경우

새로운 이름을 보게 되면, 딕셔너리에 새 원소로 집어넣어야 하고,
두 번째 혹은 그 이상 이름을 보게 되는 경우라면, 딕셔너리에서 해당
이름에 대응되는 값에 1을 더함

```
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}`



딕셔너리의 `get` 메서드

키가 이미 딕셔너리에 있는지 확인하고
키가 없다면 기본값으로 설정하는
확인 패턴은 아주 많이 쓰여서
이 일을 하는 `get()`이라는 메서드가
존재함

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

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

키가 딕셔너리에 없는 경우
기본값으로 여김 (Traceback 에러
없음)

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

get()을 이용한 간소화된 숫자 세기

키가 아직 사전에 없으면 `get()`을 사용하여 기본값으로 0을 줄 수 있음 - 그리고 1을 더함

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

기본값



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

get()을 이용한 간소화된 숫자 세기

```
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>

텍스트에서 단어 수 세기

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.

카운팅 패턴

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

```
words = line.split()
```

```
print('Words:', words)
```

```
print('Counting...')
```

```
for word in words:
```

```
    counts[word] = counts.get(word, 0) + 1
```

```
print('Counts', counts)
```

텍스트의 한 줄 안에 있는 단어 수를 셀 때,
일반적으로 `split`으로 나누어 루프에 넣고,
단어별로 수를 추적하기 위해 `딕셔너리`를
사용


```
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()
line = input('Enter a line of text:')
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}

유한 루프와 딕셔너리

딕셔너리 안에는 저장되는 순서가 없다고 해도, **for** 문을 작성하여 딕셔너리의 모든 원소를 돌 수 있음 - 딕셔너리의 모든 키를 거쳐가며 값을 찾음


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

키와 값 목록 검색

딕셔너리에서
키나 값이나 아이템
(키와 값 쌍)의
목록을 얻을 수 있음

```
>>> 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...



보너스: 두 개의 반복 변수!

- *두 개*의 반복 변수를 사용하여 딕셔너리의 키-값 쌍을 반복해서 다룸

```
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

- 매번 반복할 때,
첫 번째 변수는 키를,
두 번째 변수는 키에
대응하는 값을 나타냄

```
name = input('Enter file:')
handle = open(name)

counts = dict()
for line in handle:
    words = line.split()
    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

두 개의 루프를 중첩하여
사용

요약

- 컬렉션이란?
- 리스트 vs 딕셔너리
- 딕셔너리 상수
- 가장 많이 나온 단어
- `get()` 메서드 사용하기
- 해싱, 순서의 부재
- 딕셔너리 루프 작성
- 맛보기: 튜플
- 딕셔너리 정렬



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

Contributor:

- 이승준
- 커넥트재단

Translation:

- Hyunjun Kim
- Jeungmin Oh (tangza@gmail.com)