

Python - Dictionaries

What is a Collection?

- A collection is a great way to put more than one value in them and carry them all around in one convenient package.
- We have a bunch of values in a single “variable”

What is **not** a “Collection”

-- Most of our **variables** have one value in them - when we put a new value in the **variable** - the old value is over written

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


Dictionaries

-- Lists **index** their entries based on the position in the list

-- **Dictionaries** are like bags
- no order

-- **Index** the things to put in **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]
```

List

Key	Value
[0]	21
[1]	183

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

Dictionary

Key	Value
['course']	183
['age']	21

Dictionary Literals (Constants)

-- Dictionary literals use curly braces and have a list of **key** : **value** pairs

-- 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
{}
>>>
```

Many Counters with a Dictionary

-- A common use of a 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}
```

Dictionary Tracebacks

-- An **error** will display to reference a key which is not in the dictionary

-- 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

-- A new name is added in the **dictionary** and if this the second or later 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

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

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

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

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

Simplified counting with `get()`

-- 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
```



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 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

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!

-- Loop through the **key-value** pairs in a dictionary using **two** iteration variables

-- Each iteration, the first variable is the **key** and the 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