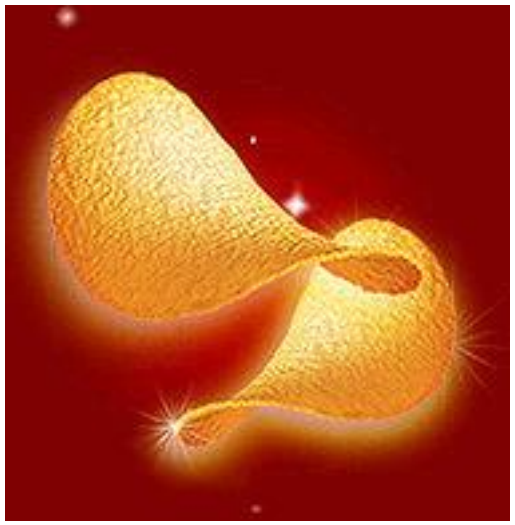
Dictionaries

Lecture 05.02

By Marina Barsky

Two ways to store things

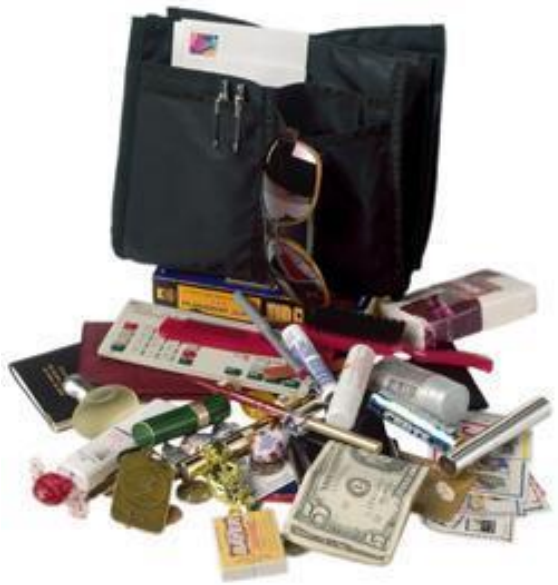


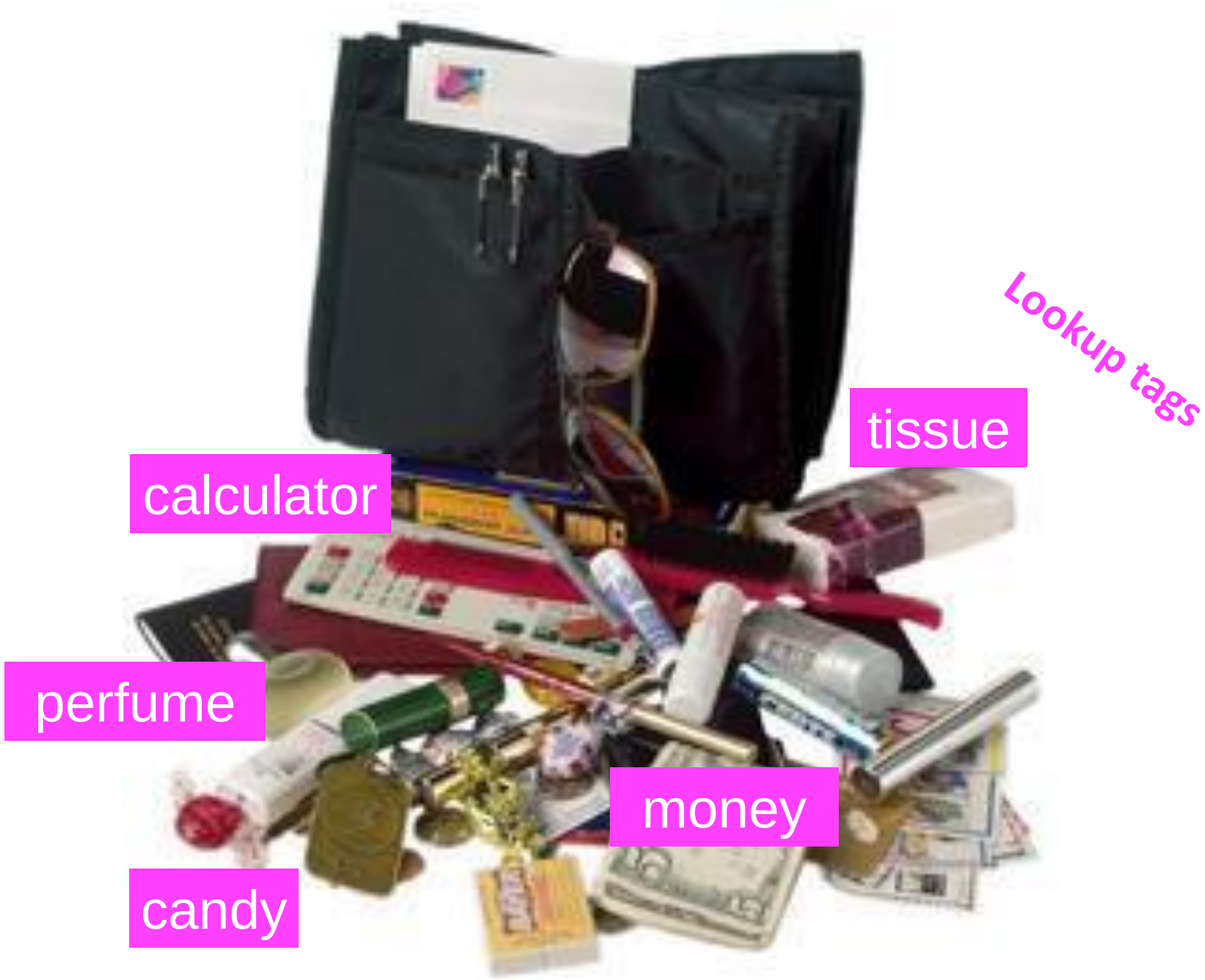
- List

 - A linear collection of values that stay in order

- Dictionary

 - A “bag” of values, each with its own label (key)




Lookup tags
tissue
calculator
perfume
money
candy

Dictionary type

- In Python a dictionary is a **set** of key - value pairs

- Also called an *Associative List* - each *value* is associated with a particular *key*

Python dictionaries

Dragon	Feb 17, **1988** – Feb 05, 1989
Snake	Feb 06, **1989** – Jan 26, 1990
Horse	Jan 27, **1990** – Feb 14, 1991
Sheep	Feb 15, **1991** – Feb 03, 1992
Monkey	Feb 04, **1992** – Jan 22, 1993
Rooster	Jan 23, **1993** – Feb 09, 1994

```
>>> d = {}
```
creates an empty dictionary, `d`

```
>>> d[1992] = 'monkey'
```
`1992` is the key
`'monkey'` is the value

```
>>> d[1993] = 'rooster'
```
`1993` is the key
`'rooster'` is the value

```
>>> d
{1992:'monkey', 1993:'rooster'}
```
Curly! And colony!

```
>>> d[1992]
'monkey'
>>> d[1969]
key error
```

The keys are unique

```
>>> d = {}
>>> d[1992] = 'ape'
>>> d[1992] = 'monkey'
>>> d
{1992:'monkey'}
```

creates an empty dictionary, `d`

`'monkey'` overrides value for key 1992

Only the last value is stored

Familiar behavior

```
>>> d = {1992:'monkey', 1993:'rooster'}
>>> 1992 in d
True
>>> 1969 in d
False
```

`in` checks if a key is present

```
>>> len(d)
2
```

`len` returns the # of key/value pairs.

in operator only checks the **keys** of the dictionary, not the values
This is very similar to real dictionaries – you ask if the word is in the dictionary meaning only a keyword, not the translation or explanation

Contest winners: solution

```
f = open("scores.txt")

sorted_scores = []
dict_names = {}
for line in f:
    line_arr = line.split()
    name = line_arr[0]
    score = float(line_arr[1])

    sorted_scores.append(score)
    dict_names[score] = name


sorted_scores.sort()
sorted_scores.reverse()
template = "{} is in place {} with score {}"
for i in range(3):
    score = sorted_scores[i]
    name = dict_names[score]
    print(template.format(name,i+1,score))
```

Lists inside dictionaries

```
>>> d = {1992:'monkey', 1993:'rooster'}
```

```
>>> list(d.keys())
[ 1992, 1993 ]
```

`d.keys` returns a "view object" of all keys, convert it to a list!

```
>>> list(d.values())
[ 'monkey', 'rooster' ]
```

`d.values` returns a "view object" of all values, convert it to a list!

```
>>> list(d.items())
[ (1992, 'monkey'), (1993, 'rooster') ]
```

`d.items()` returns a "view object" of all key, value ***pairs*** *(tuples).*

For loops over dictionaries

```
for key,value in dict.items():
    print (key, '=', value)
```

Note: dictionaries are unsorted so the output could be in any order.

Sorted keys

```
keys = list(dict.keys())
keys.sort()
for key in keys:
    print (key, '=', dict[key])
```

```
def vc( filename ):
    """vocabulary counting program """
    f = open( filename )
    text = f.read()
    f.close()

    words = text.split()
    print("There are", len(words), "words.")


    d = {}

    for w in words:

        if w not in d:
            d[w] = 1
        else:
            d[w] += 1


    print ("There are", len(d), "distinct words.\n")

    return len(d)    # return the number of distinct words
```

What if we wanted the number of ***different*** words in the file? This would be the author's ***vocabulary size***, instead of the total word count.

Simplified counting with get()

The 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 dictionary method called *get()* that does this for us:

```
d = {}

for w in words:

    d[w] = d.get(w, 0) + 1

print ("There are", len(d), "distinct words.\n")

return len(d)    # return the number of distinct words
```

Get previous count for w, zero if not in dictionary

Vocabularists

Shakespeare used ***31,534 different words*** and a grand total of 884,647 words counting repetitions (across his works)

http://www-math.cudenver.edu/~wbriggs/qr/shakespeare.html

Active vocabulary estimates range from 10,000-60,000.
Passive vocabulary estimates are much higher.

Many Shakespearean contributions:

Adjectives:		Shakespeare
aerial	auspicious	baseless
beached	bloodstained	blushing
circumstantial	consanguineous	deafening
disgraceful	domineering	enrapt
epileptic	equivocal	eventful
fashionable	foregone	frugal
generous	gloomy	gnarled

One contemporary author in the Oxford Eng. Dictionary…

which word?

Vocabularists

Shakespeare used ***31,534 different words*** and an estimated vocabulary of 66,534 words (across his works)

http://kottke.org/10/04/how-many-words-did-shakespeare-know

Average English speaker knows 10,000 to 20,000 words.

Many Shakespearean contributions:

Adjectives:

aerial	auspicious	baseless
beached	bloodstained	blushing
circumstantial	consanguineous	deafening
disgraceful	domineering	enrapt
epileptic	equivocal	eventful
fashionable	foregone	frugal
generous	gloomy	gnarled

Check this out

'Muggle' goes into Oxford English Dictionary

JK Rowling's word for non-wizards - "muggle" - has made it into the new edition of the Oxford English Dictionary (OED).

The draft definition according to the dictionary's website says:

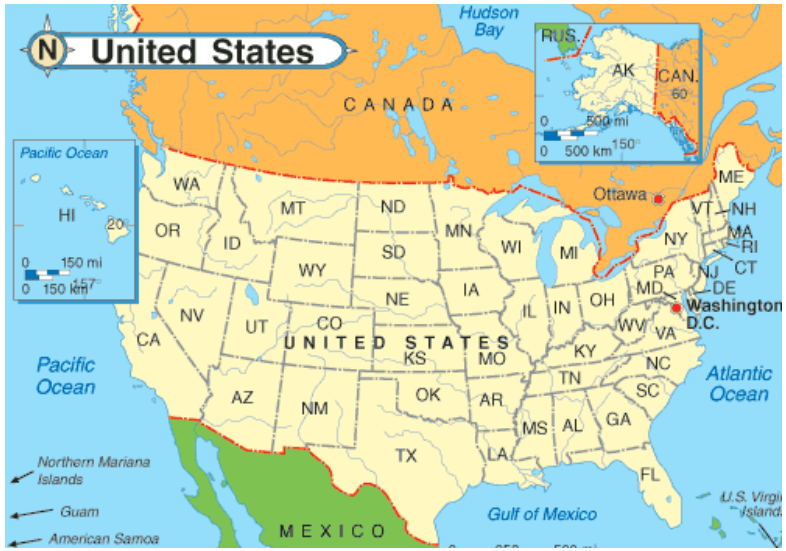
- ***Muggle:*** *invented by JK (Joanne Kathleen) Rowling (b. 1965), British author of children's fantasy fiction (see quot. 1997).*

  In the fiction of JK Rowling: a person who possesses no magical powers. Hence in allusive and extended uses: a person who lacks a particular skill or skills, or who is regarded as inferior in some way.

J. K. Rowling

Sample program: A state challenge…

Consider a game of guessing all 50 states…




```
state_list = ['AL', 'AK', 'AZ', 'AR', 'CA', 'CO', 'CT', 'DE',
              'DC', 'FL', 'GA', 'HI', 'ID', 'IL', 'IN', 'IA',
              'KS', 'KY', 'LA', 'ME', 'MD', 'MA', 'MI', 'MN',
              'MS', 'MO', 'MT', 'NE', 'NV', 'NH', 'NJ', 'NM',
              'NY', 'NC', 'ND', 'OH', 'OK', 'OR', 'PA', 'RI',
              'SC', 'SD', 'TN', 'TX', 'UT', 'VT', 'VA', 'WA',
              'WV', 'WI', 'WY']
for state_name in state_list:
    state_dict [state_name] = 0
```

```
stateDict = {'AL': 0, 'AK': 0, 'AZ': 0, 'AR': 0, 'CA': 0, 'CO': 0, 'CT': 0,
             'DE': 0, 'DC': 0, 'FL': 0, 'GA': 0, 'HI': 0, 'ID': 0, 'IL': 0,
             'IN': 0, 'IA': 0, 'KS': 0, 'KY': 0, 'LA': 0, 'ME': 0, 'MD': 0,
             'MA': 0, 'MI': 0, 'MN': 0, 'MS': 0, 'MO': 0, 'MT': 0, 'NE': 0,
             'NV': 0, 'NH': 0, 'NJ': 0, 'NM': 0, 'NY': 0, 'NC': 0, 'ND': 0,
             'OH': 0, 'OK': 0, 'OR': 0, 'PA': 0, 'RI': 0, 'SC': 0, 'SD': 0,
             'TN': 0, 'TX': 0, 'UT': 0, 'VT': 0, 'VA': 0, 'WA': 0, 'WV': 0,
             'WI': 0, 'WY': 0}
```

```
def state_challenge( state_dict ):

    while 0 in state_dict.values():
        print(list(state_dict.values()).count(0), \
              "states left to guess")
        guess = input("Name a state: ")
        if guess not in state_dict:
            print 'Try again...'
        elif state_dict[guess] == 0:
            print 'Yes!'
            state_dict[guess] += 1
        else:
            print('Already guessed...')
            state_dict[guess] += 1

    print('Phew!')
```

Tell user how many left to guess

wasn't a key at all

newly guessed key: value was 0

repeated guess

Tuples and Dictionaries

The *items()* method in dictionaries returns a read-only sequence of (key, value) *tuples*

```
>>> d = dict()
>>> d['csev'] = 2
>>> d['cwen'] = 4
>>> for (k,v) in d.items():
...     print(k, v)
...
csev 2
cwen 4
>>> tups = d.items()
>>> print(tups)
dict_items([('csev', 2), ('cwen', 4)])
```

Tuples are Comparable

The comparison operators work with tuples and other sequences. If the first item is equal, Python goes on to the next element, and so on, until it finds elements that differ.

```
>>> (0, 1, 2) < (5, 1, 2)
True
>>> (0, 1, 2000000) < (0, 3, 4)
True
>>> ( 'Jones', 'Sally' ) < ('Jones', 'Sam')
True
>>> ( 'Jones', 'Sally') > ('Adams', 'Sam')
True
```

Sorting Lists of Tuples

We can take advantage of the ability to sort a list of tuples to get a sorted version of a dictionary

Keys come first, so we sort the dictionary by the key using the items() method and sorted() function

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> d.items()
dict_items([('a', 10), ('c', 22), ('b', 1)])
>>> sorted(d.items())
[('a', 10), ('b', 1), ('c', 22)]
```

Using sorted()

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> t = sorted(d.items())
>>> t
[('a', 10), ('b', 1), ('c', 22)]
>>> for k, v in sorted(d.items()):
...     print(k, v)
...
a 10
b 1
c 22
```

Sort by Values Instead of Key

If we could construct a list of tuples of the form (value, key) we could sort by value

We do this with a *for* loop that creates a list of tuples

```
>>> c = {'a':10, 'b':1, 'c':22}
>>> tmp = list()
>>> for k, v in c.items() :
      tmp.append( (v, k) )
>>> print(tmp)
[(10, 'a'), (1, 'b'), (22, 'c')]
>>> tmp = sorted(tmp, reverse=True)
>>> print (tmp)
[(22, 'c'), (10, 'a'), (1, 'b')]
```

Even Shorter Version

```
>>> c = {'a':10, 'b':1, 'c':22}

>>> print( sorted( [ (v,k) for k,v in c.items() ] ) )

[(1, 'b'), (10, 'a'), (22, 'c')]
```

List comprehension creates a list of value,key pairs.
In this case, we make a list of reversed tuples and then sort it.

http://wiki.python.org/moin/HowTo/Sorting

The top 10 most common words

```
fhand = open('romeo.txt')
counts = dict()
for line in fhand:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0 ) + 1

lst = list()
for key, val in counts.items():
    newtup = (val, key)
    lst.append(newtup)

lst = sorted(lst, reverse=True)

for val, key in lst[:10]:
    print(key, val)
```