Tuples, Dictionaries & Files

Content adopted from ITC, Netherland

Tuple

- We introduce a new data structure.
- **Tuples** are *very much* like Lists!

```
>>> lst = ['a', 'b', 'c', 'd', 'e']    # List
>>> lst[1]
'b'

>>> tpl = ('a', 'b', 'c', 'd', 'e')    # Tuple
>>> tpl[1]
'b'
```

Parentheses?

- Parentheses '(' and ')' are also used for grouping computations.
- There may be confusion with Tuples!
- Is de result of (1+1) the integer 2 or the Tuple (2)?

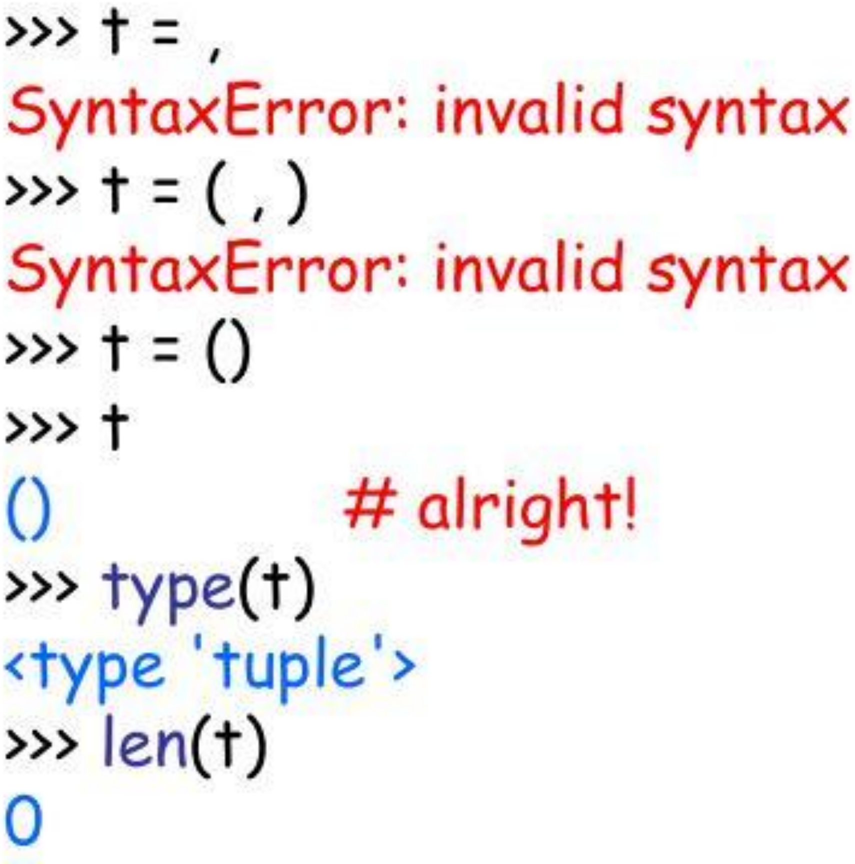
Use a ',' to create a 1-element tuple:

```
>>> t1 = ('a',)
>>> type(t1)
<type 'tuple'>
```

Python also uses this convention:

```
>>> t1
('a',)
```

The Empty Tuple

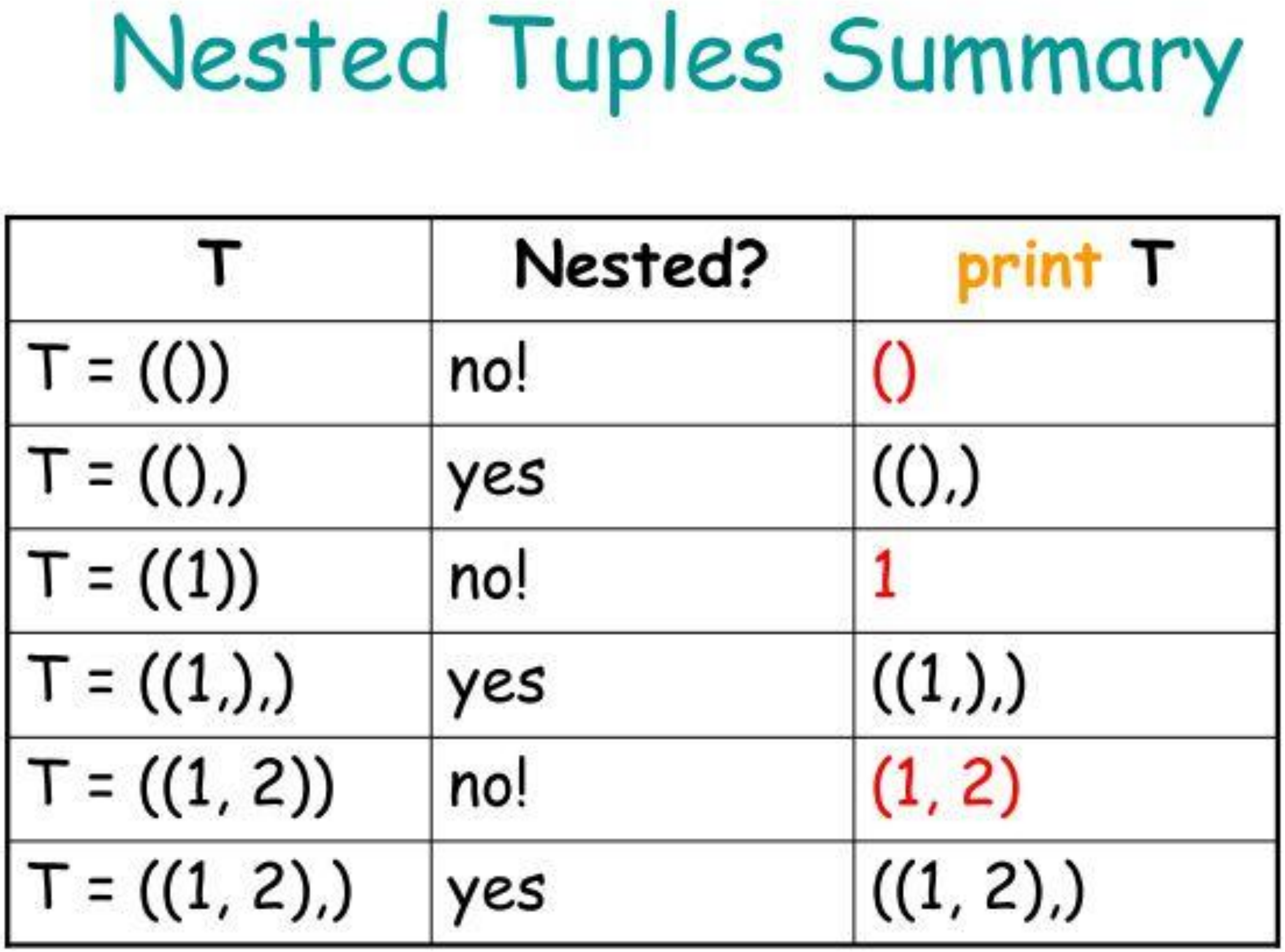


```
>>> t = ,
SyntaxError: invalid syntax
>>> t = ( , )
SyntaxError: invalid syntax
>>> t = ()
>>> t
()                # alright!
>>> type(t)
<type 'tuple'>
>>> len(t)
0
```

Tuple Summary

Do you like (tuple,) soup?

T	Tuple?	print T
T = ()	yes	()
T = (1)	no!	1
T = (1,)	yes	(1,)
T = (1, 2)	yes	(1, 2)



Nested Tuples Summary

T	Nested?	print T
T = (())	no!	()
T = ((),)	yes	((),)
T = ((1))	no!	1
T = ((1,),)	yes	((1,),)
T = ((1, 2))	no!	(1, 2)
T = ((1, 2),)	yes	((1, 2),)

Complicated?

Not really. But it's easy to make mistakes

Here's a trick:

- Define all tuples with (... ,)

Unfortunately, there's one exception:

```
>>> ( ,)
SyntaxError: invalid syntax
```

Lists can change

```
>>> lst = ['a', 'b', 'c']
>>> lst[0]
'a'
>>> lst[0] = 'd'
>>> lst
['d', 'b', 'c']
```

Tuples can NOT change!

```
>>> tpl = ('a', 'b', 'c')
>>> tpl[0]
'a'

>>> tpl[0] = 'd'
TypeError: object does not support item assignment

>>> del(tpl[1])
TypeError: object doesn't support item deletion
```

Tuples are Immutable

- Lists are **mutable**, i.e. they can change
- Tuples – like strings! - are **immutable**

What does this do?

```
>>> tpl = ('a', 'b', 'c')
>>> tpl = tpl[:1] + ('d',) + tpl[1:]
>>> tpl
('a', 'd', 'b', 'c')
```

Question

Why have **Tuples** if there are **Lists**?

Answer:

1. Because the locations of objects inside tuples don't change, tuples can be handled more efficiently than lists.
2. Tuples *aren't supposed* to change!

Tuple Assignment

A tuple can be **assigned** to another tuple:

```
>>> tuple1 = tuple2
```

The following rules apply:

1. Left side must be all variables.
2. The tuples must be of equal length.
3. Right side values are evaluated before the assignment takes place!!

Example 1

Assign 3 variables in one statement:

```
>>> a, b, c = 1, 2, 3
>>> a
1
>>> b
2
>>> c
3
```

Example 2

Swapping values:

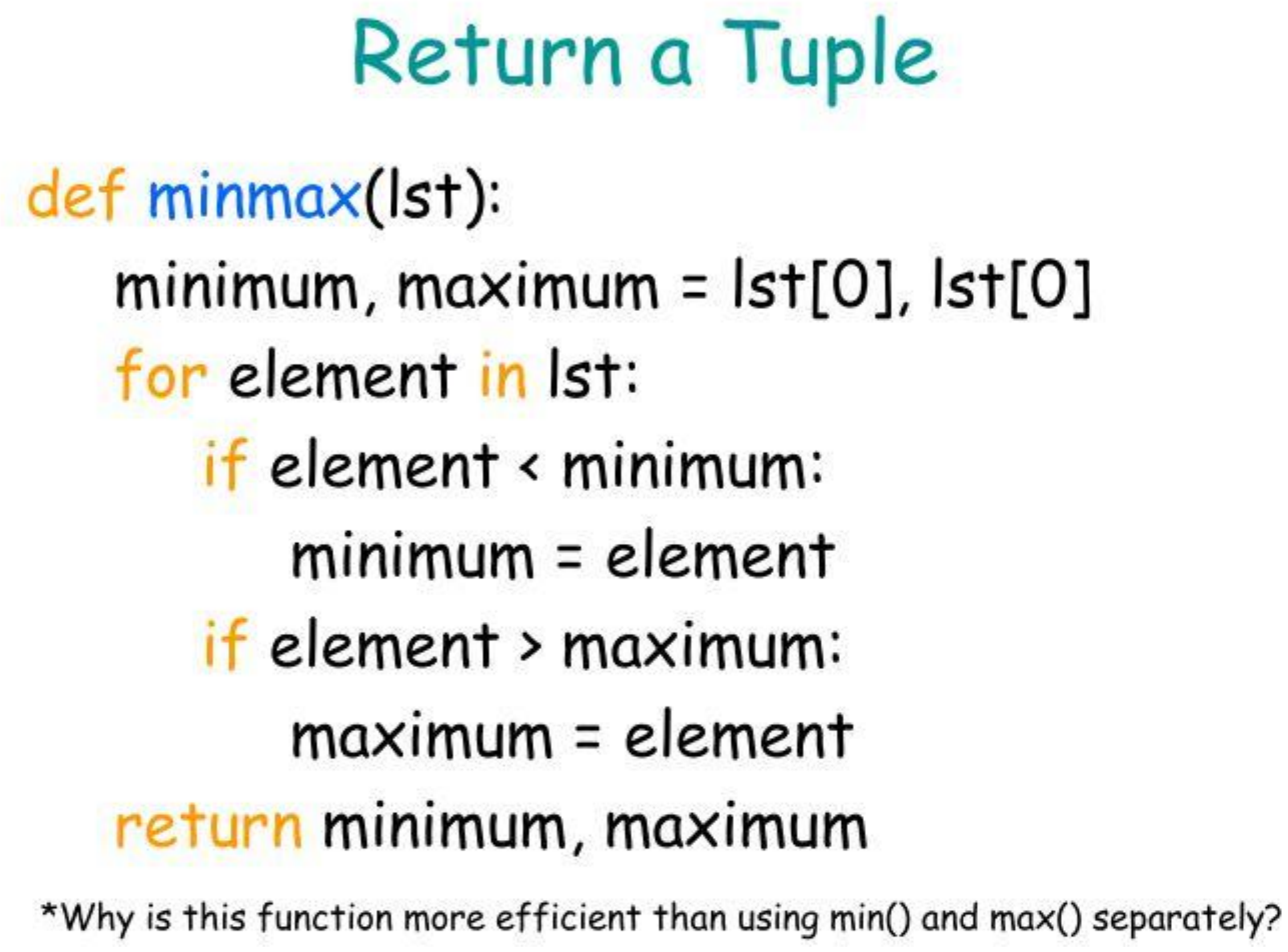
```
>>> a, b = 1, 2   # equivalent to (a,b)=(1,2)
>>> a, b
(1, 2)
>>> a, b = b, a
>>> a, b
(2, 1)
```

Question.
Is this the same as:

```
>>> a = b
>>> b = a
>>> a, b
(2, 2)        # Huh??
```

Errors

```
>>> (a, b, 3) = (4, 5, 6)
SyntaxError: can't assign to literal

>>> (a, b, c, d) = (1, 2, 3)
ValueError: need more than 3 values to unpack

>>> (a, b, c) = (1, 2, 3, 4)
ValueError: too many values to unpack
```



Return a Tuple

```
def minmax(lst):
    minimum, maximum = lst[0], lst[0]
    for element in lst:
        if element < minimum:
            minimum = element
        if element > maximum:
            maximum = element
    return minimum, maximum
```

*Why is this function more efficient than using min() and max() separately?

Min & Max

Calculate the minimum and maximum of a list of numbers:

```
>>> minmax((12, 45, 23, 89, 78, 34, 6, 78, 23))
(6, 89)
```

Why do we need the extra brackets?

```
>>> minmax(12, 45, 23, 89, 78, 34, 6, 78, 23)
TypeError: minmax() takes exactly 1 argument (9 given)
```

Sequences!

```
>>> minmax((12, 45, 23, 89, 34, 6, 78, 23))
(6, 89)
>>> minmax([84,76,25,44])
(25, 84)
>>> minmax('hello')
('e', 'o')
```

Lists, Strings and Tuples are **Sequences**!

Chapter 11
Dictionaries

Dictionaries

- New data structure called **dictionary**
- Instead of an *index*, a **key** is used to store a **value**.

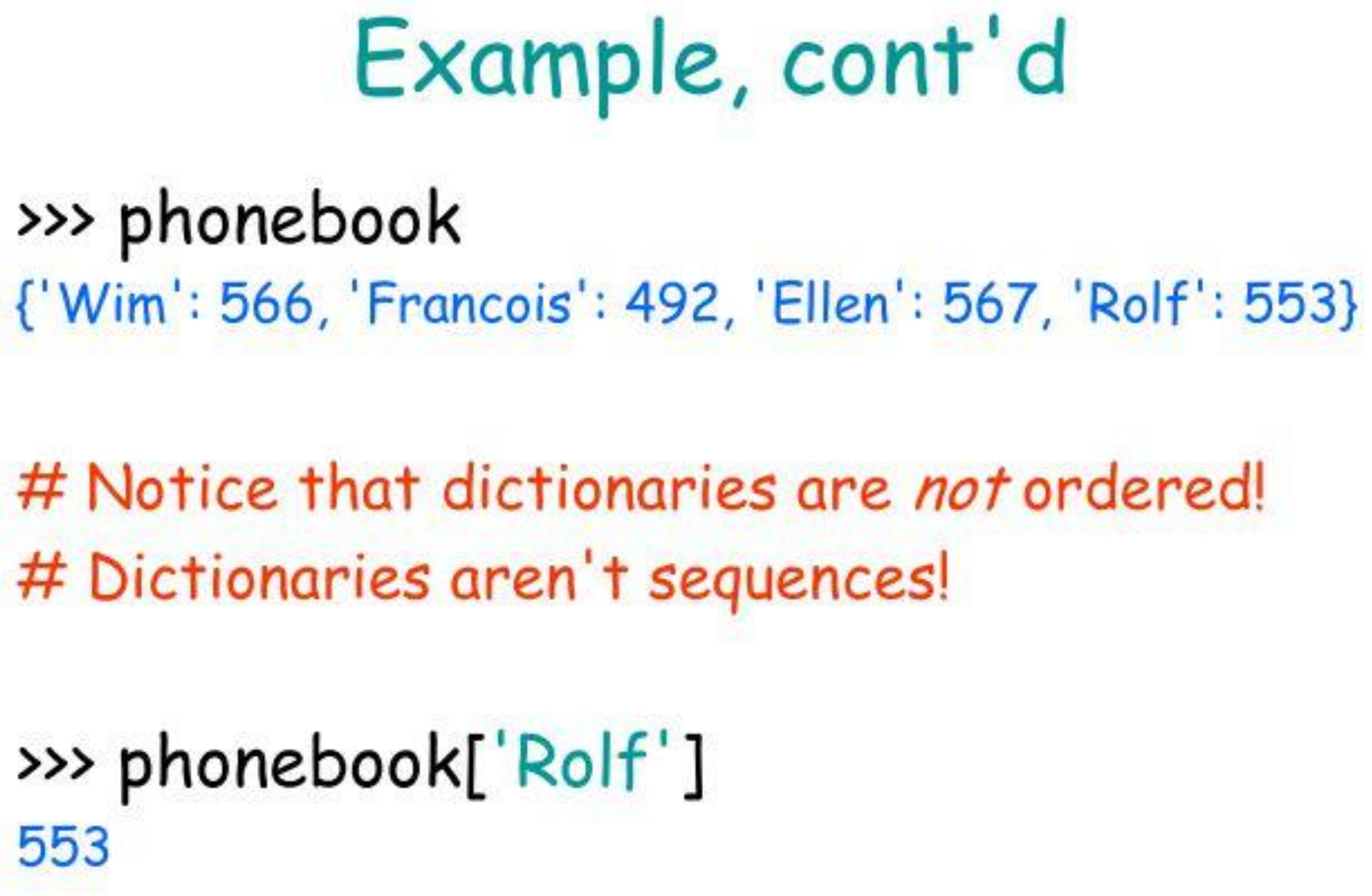
- Dictionaries are handy for storing things that are known by name rather than by location.

- Dictionaries are defined with '{' and '}'

Example

- The telephone book of the department

```
phonebook={}                    # empty dict.
phonebook['Rolf']=553
phonebook['Wim']=566
phonebook['Francois']=492
phonebook['Ellen']=567
```



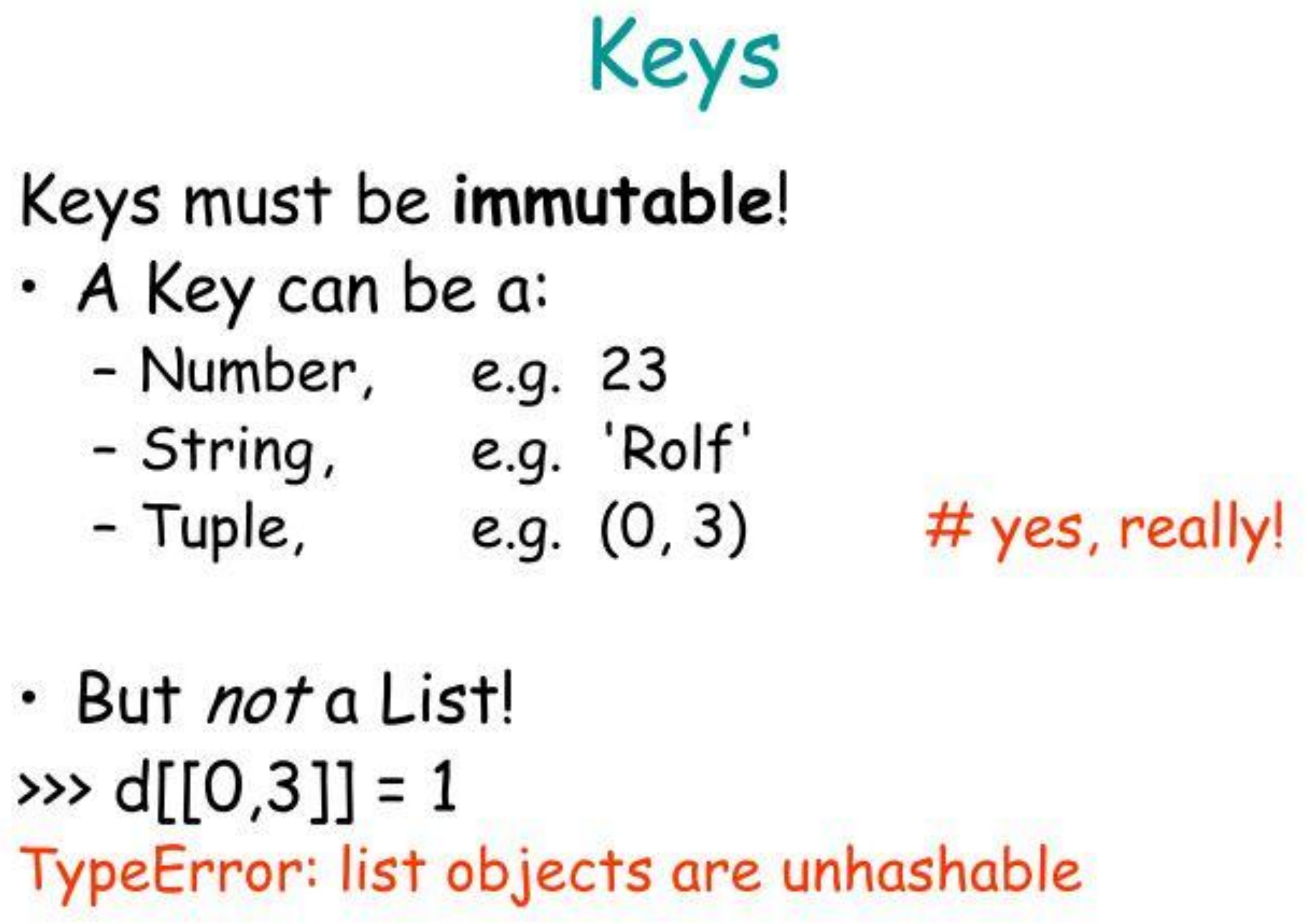
Example, cont'd

```
>>> phonebook
{'Wim': 566, 'Francois': 492, 'Ellen': 567, 'Rolf': 553}

# Notice that dictionaries are not ordered!
# Dictionaries aren't sequences!

>>> phonebook['Rolf']
553
```

Key:Value Pairs

- In other words, a dictionary stores **key:value pairs**.
- We could also have done like this:

```
phonebook = {'Wim': 566, 'Francois': 492,
   'Ellen': 567, 'Rolf': 553}
```



Keys

Keys must be **immutable**!

- A Key can be a:
  - Number, e.g. 23
  - String, e.g. 'Rolf'
  - Tuple, e.g. (0, 3) # yes, really!

- But *not* a List!

```
>>> d[[0,3]] = 1
TypeError: list objects are unhashable
```

Values

- Values can be **anything**!
- For instance, if Rolf has 2 phones, then we *may* store that fact as a tuple containing two phone numbers:

```
>>> phonebook['Rolf'] = (553, 567)
>>> phonebook['Rolf']
(553, 567)
```

Dictionary Methods

Functions of Objects are called **Methods**.

- Syntax: *object.method()*

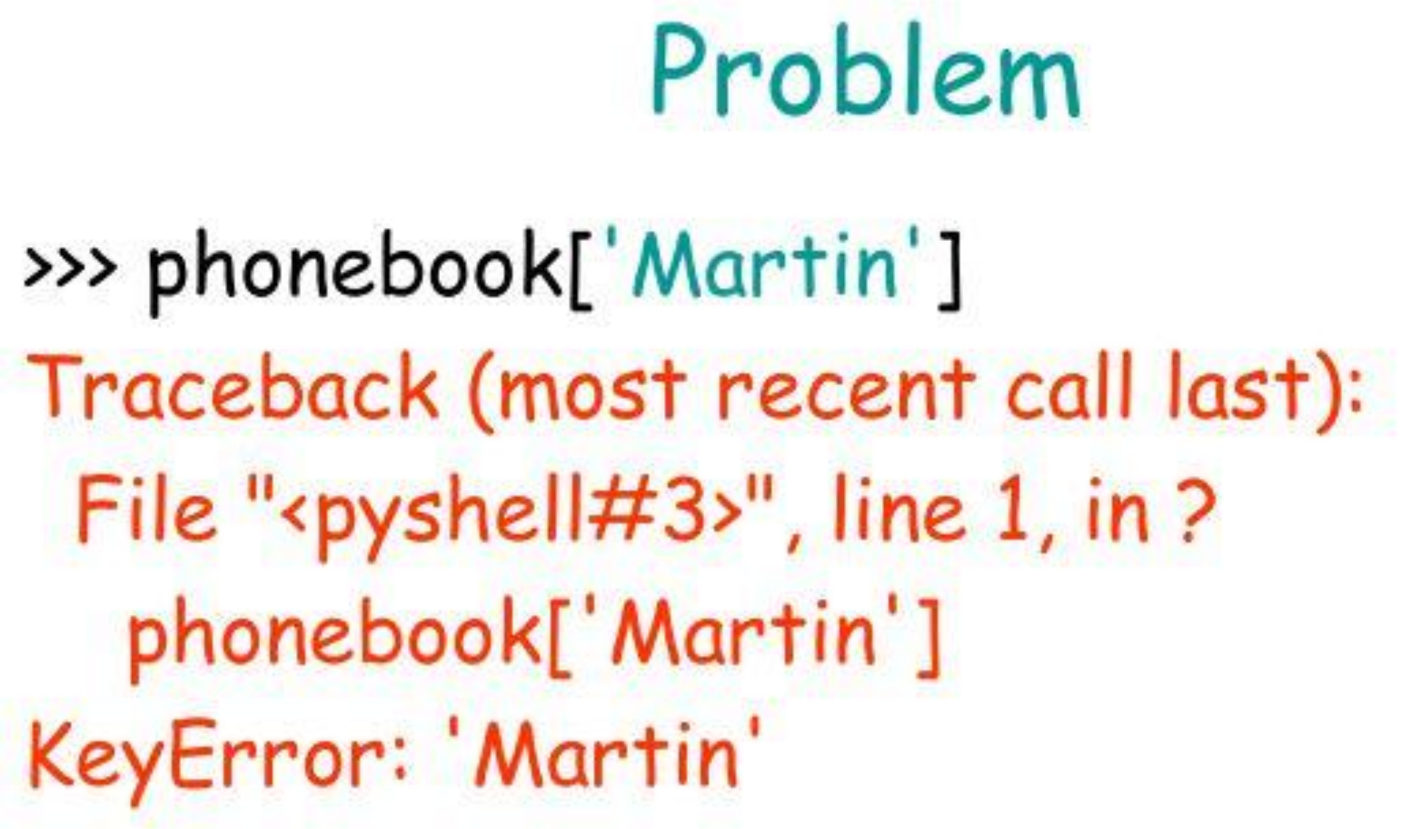
Some dictionary methods:

- keys(), returns a list of keys.
- values(), returns a list of values.
- items(), returns the key:value pairs as a list of tuples:

  [(key, value), (key, value), ...]

Example

```
>>> phonebook
{'Wim': 566, 'Francois': 492, 'Ellen': 567, 'Rolf': (553, 567)}
>>> phonebook.keys()
['Wim', 'Francois', 'Ellen', 'Rolf']
>>> phonebook.values()
[566, 492, 567, (553, 567)]
>>> phonebook.items()
[('Wim', 566), ('Francois', 492), ('Ellen', 567), ('Rolf', (553, 567))]
```



Problem

```
>>> phonebook['Martin']
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in ?
    phonebook['Martin']
KeyError: 'Martin'
```

The get() Method

```
>>> phonebook.get('Martin', 9)
9
```

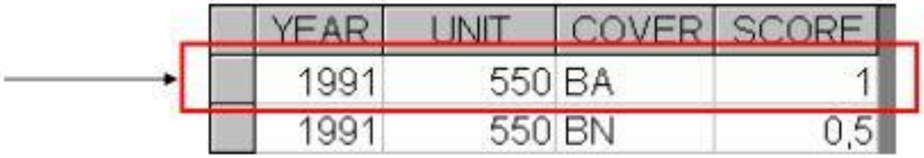
Too bad about the syntax!*

```
phonebook['Martin']
```

*This may change in Python 3000

Dictionary & DB

In a connection to a database, one record may be returned as a dictionary, containing field:value pairs of the record:



	YEAR	UNIT	COVER	SCORE
	1991	550	BA	1
	1991	550	BN	0,5

```
record = {'YEAR':1991, 'UNIT':550, 'COVER':'BA', 'SCORE':1}

>>> print record['UNIT'], record['COVER']
550  BA
```

Chapter 14
Files

Files

Files are used for

storing data permanently on:

- Hard disk
- CD-ROM
- Pen drive
- Floppy disk
- Tape

Directories

Files are organized into directories

Directories can be specified in 2 ways:

1. By using the absolute path
   - e.g. "C:\Wim\Education\Python-2005"
2. By using the relative path
   - e.g. "..\..\Scratch\Photos"

Current Directory

- When a program runs, it always has a Current Working Directory (CWD).
- All the relative paths are taken with the CWD as a starting point.

- A filename *without* a directory part is assumed to be in the CWD!
- Usually, the CWD is the directory where the program was started from.

CWD, example

For instance, if the CWD is:
"C:\Wim\Education\Python-2005"

then the relative path:
"..\..\Python"

brings me to the absolute path:
"C:\Wim\Python"

Filenames

A filename consists of 2 parts:

1. a directory name (optional)
2. a base name (mandatory)

Everything *before* the last '\' is the directory name.

Everything *after* the last '\' is the base name.

Dirname & Basename

1. No directory:

- "Redpoll. jpg"

2. Relative path:

- "..\..\..\Temp\Redpoll.jpg"

3. Absolute path:

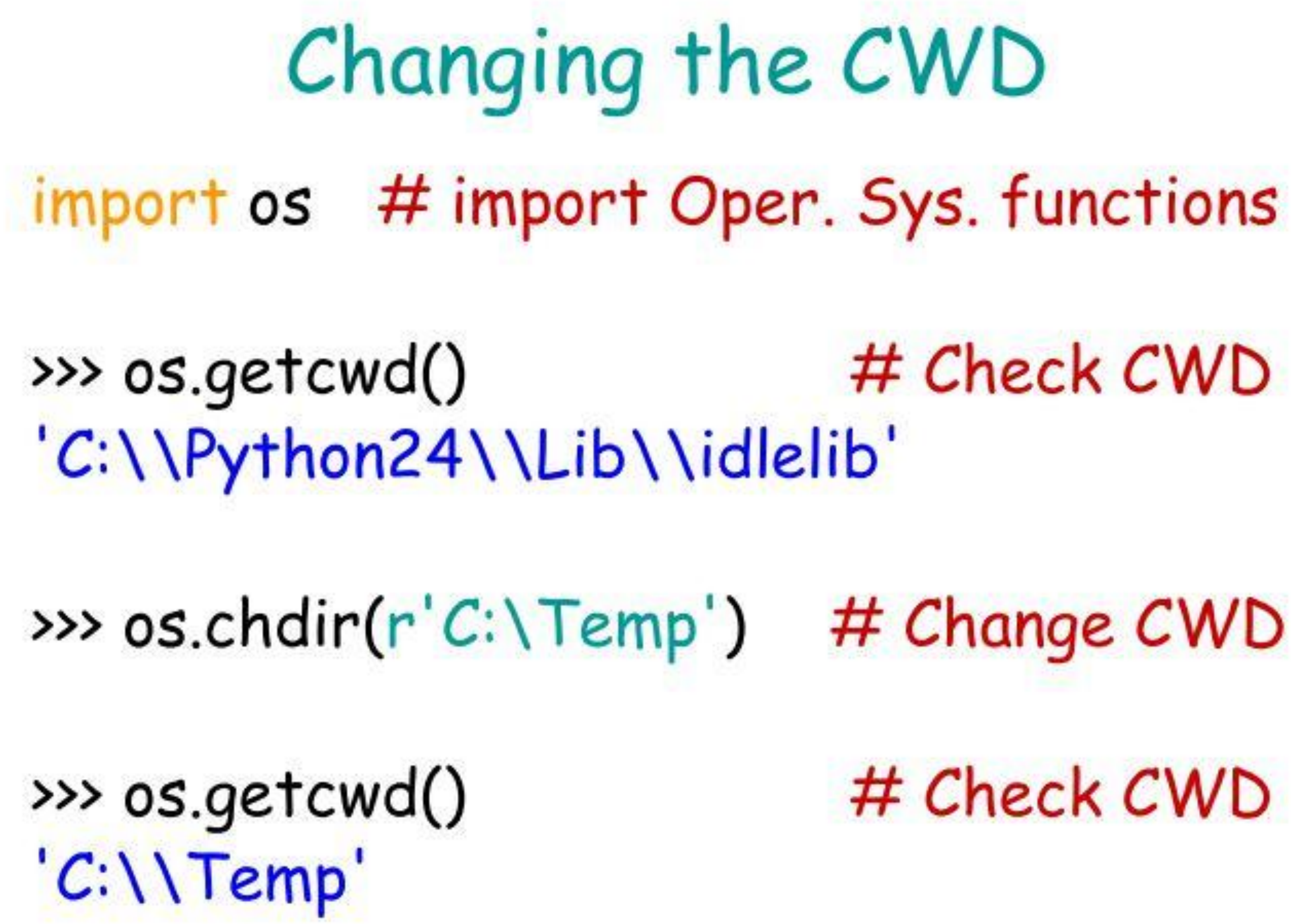
- "C:\Wim\Birds\Pictures\Redpoll.jpg"

Examples

Assume that the CWD is
"C:\Wim\Birds\Pictures"

Then:

1. "Redpoll. jpg" =
   "C:\Wim\Birds\Pictures\Redpoll.jpg"
2. "..\..\..\Temp\Redpoll.jpg" =
   "C:\Temp\Redpoll.jpg"
3. "C:\Wim\Birds\Pictures\Redpoll.jpg" =
   "C:\Wim\Birds\Pictures\Redpoll.jpg"



Changing the CWD

```
import os   # import Oper. Sys. functions

>>> os.getcwd()                # Check CWD
'C:\\Python24\\Lib\\idlelib'

>>> os.chdir(r'C:\Temp')    # Change CWD

>>> os.getcwd()                # Check CWD
'C:\\Temp'
```

2 options

Typically, there are 2 options used for specifying file names:

1. Use absolute paths for file names.

2. Change the current working directory, and use relative paths for file names.

Files are like books

- To use a book you have to open it.
- You can read from it, or write in it.
- When you're done, you have to close it.

- Most of the time you read the book from begin to end,
- but you can also skip around.

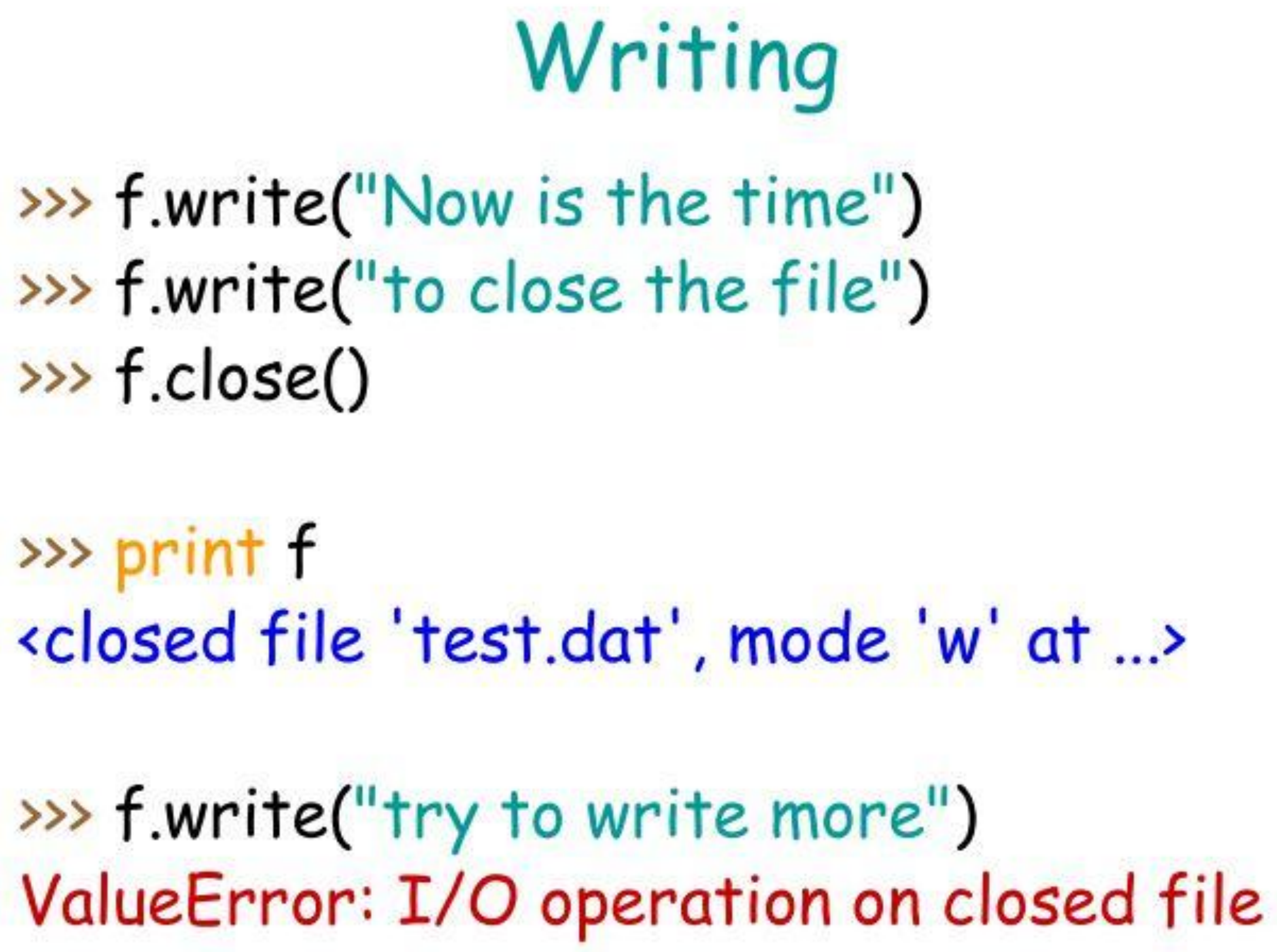
File Objects

When you open a file, a file object is created.

All actions on the file run via this object

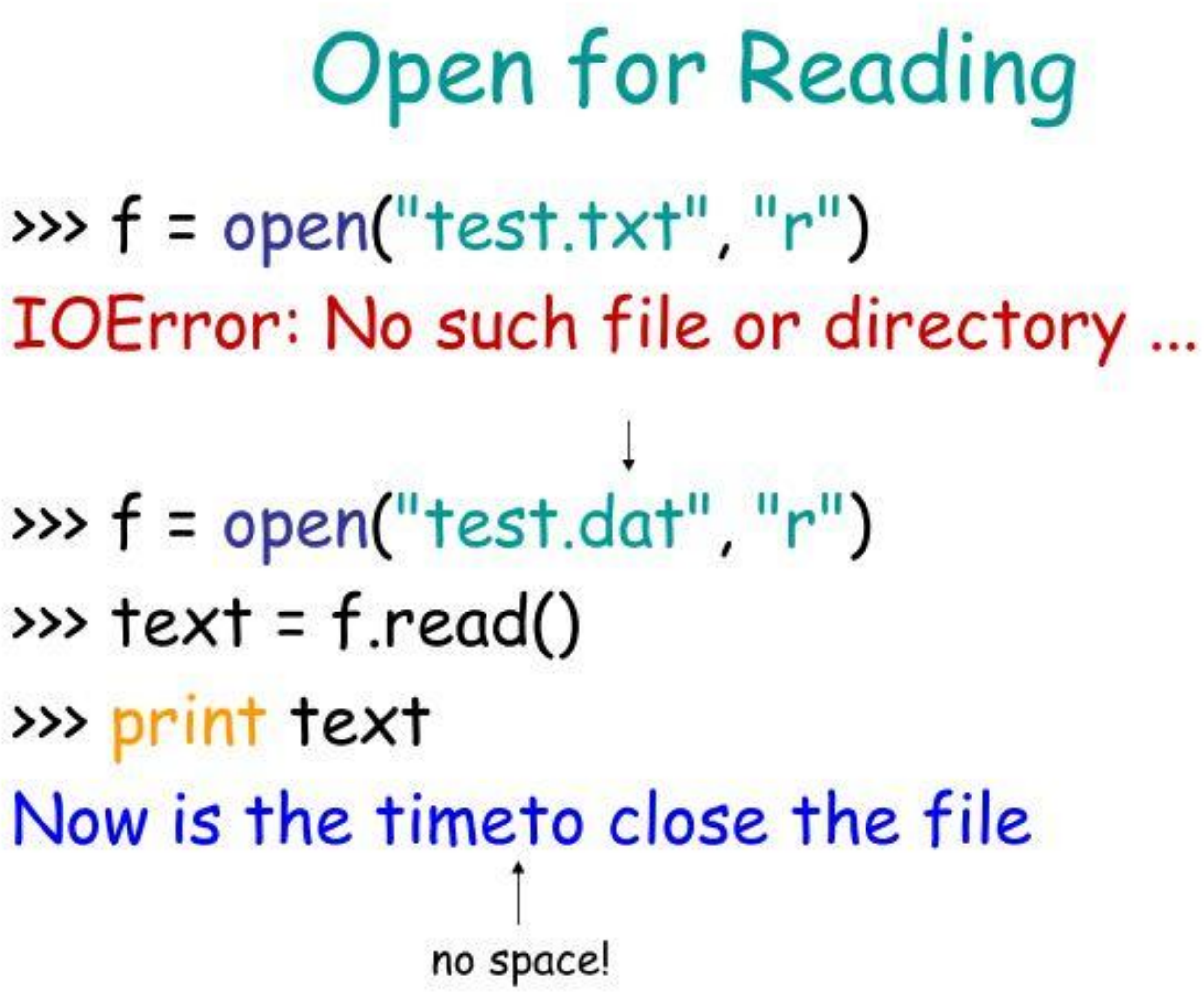
```
>>> f = open("test.dat", "w")
>>> print f
<open file 'test.dat', mode 'w' at ...>
```

If "test.dat" exists it will be destroyed!!



Writing

```
>>> f.write("Now is the time")
>>> f.write("to close the file")
>>> f.close()

>>> print f
<closed file 'test.dat', mode 'w' at ...>

>>> f.write("try to write more")
ValueError: I/O operation on closed file
```



Open for Reading

```
>>> f = open("test.txt", "r")
IOError: No such file or directory ...

>>> f = open("test.dat", "r")
>>> text = f.read()
>>> print text
Now is the timeto close the file
```

no space!

Read(n)

```
>>> f = open("test.dat", "r")

>>> text = f.read(10)
>>> print text
Now is the
```

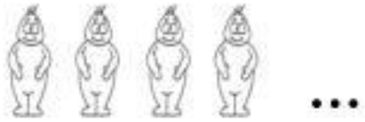
Successive reads

```
>>> f.read(9)              # read 9 chars
' timeto c'
>>> f.read(1000000)        # read remaining
'lose the file'
>>> f.read()               # anything else?
''
```

Returns '' if there are no characters left!

Copy File

```
def CopyFile(oldFile, newFile):
    f1 = open(oldFile, 'r')
    f2 = open(newFile, 'w')
    text = f1.read(1)
    while not text == '':
        f2.write(text)
        text = f1.read(1)
    f2.close()
    f1.close()
```



reads one character at the time

Copy File

```
def CopyFile(oldFile, newFile):
    f1 = open(oldFile, 'r')
    f2 = open(newFile, 'w')
    text = f1.read(1)
    while not text == '':
        f2.write(text)
        text = f1.read(1)
    f2.close()
    f1.close()
```

1. Initialization

2. Loop condition

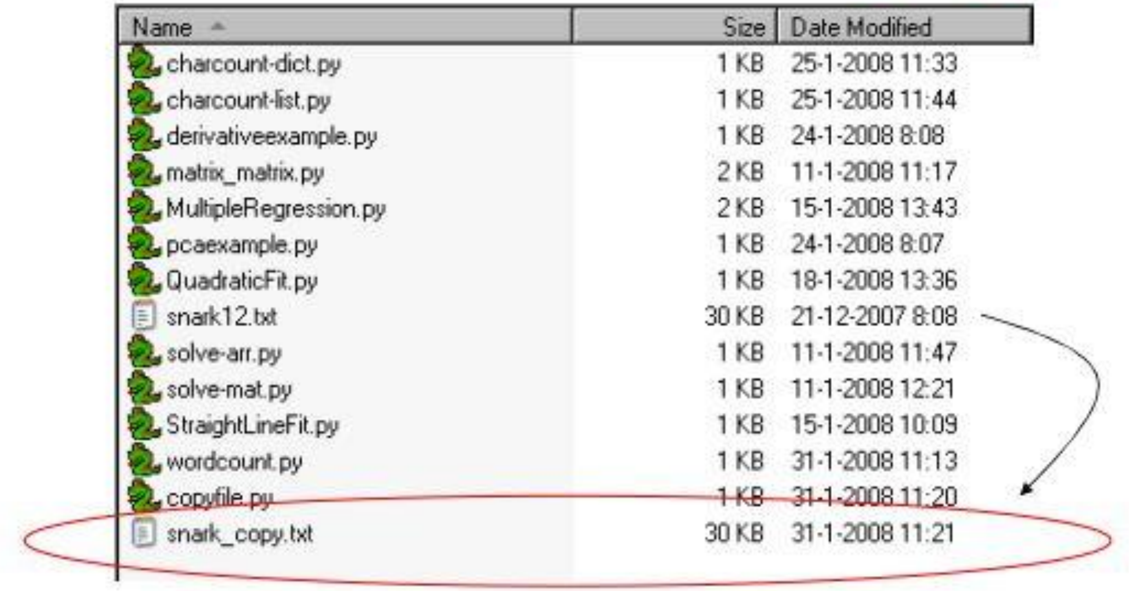
... Do the work

3. Next step

Usage

To create a copy (clone) of your file:

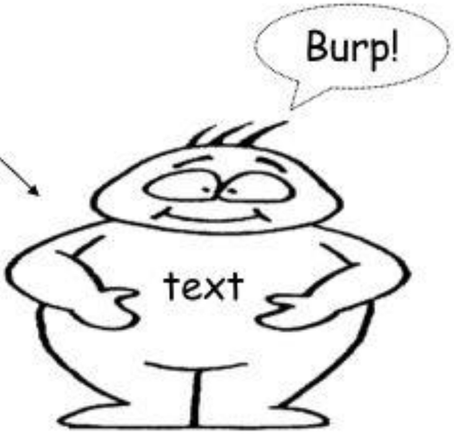
```
>> CopyFile('snark12.txt', 'snark_copy.txt')
```



Name	Size	Date Modified
charcount-dict.py	1 KB	25-1-2008 11:33
charcount-list.py	1 KB	25-1-2008 11:44
derivativeexample.py	1 KB	24-1-2008 8:08
matrix_matrix.py	2 KB	11-1-2008 11:17
MultipleRegression.py	2 KB	15-1-2008 13:43
pcaexample.py	1 KB	24-1-2008 8:07
QuadraticFit.py	1 KB	18-1-2008 13:36
snark12.txt	30 KB	21-12-2007 8:08
solve-arr.py	1 KB	11-1-2008 11:47
solve-mat.py	1 KB	11-1-2008 12:21
StraightLineFit.py	1 KB	15-1-2008 10:09
wordcount.py	1 KB	31-1-2008 11:13
copyfile.py	1 KB	31-1-2008 11:20
snark_copy.txt	30 KB	31-1-2008 11:21

Greedy Copy File

```
def GreedyCopyFile(oldFile, newFile):
    f1 = open(oldFile, 'r')
    f2 = open(newFile, 'w')
    text = f1.read()
    if not text == '':
        f2.write(text)
    f2.close()
    f1.close()
```



reads the whole file at once

Types of Files

1. Text files - Human readable

   e.g. .txt .csv .htm .xml .py

2. Binary files - Anything else!

   e.g. .doc .xls .jpg .mp3 .pyc

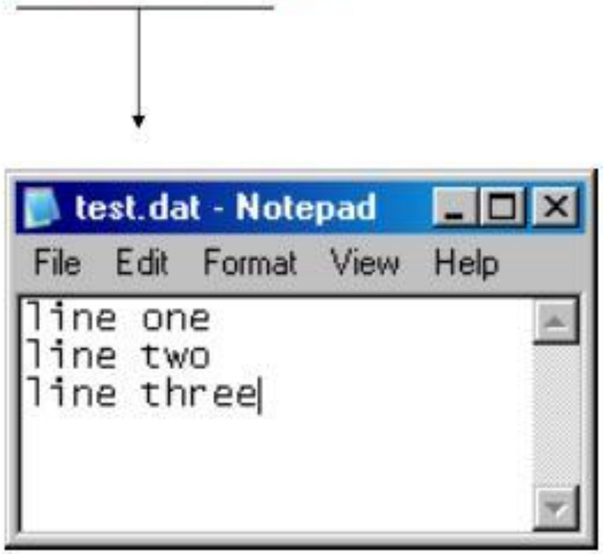
Text files

A text file contains printable characters and spaces, organized into lines separated by newline characters.

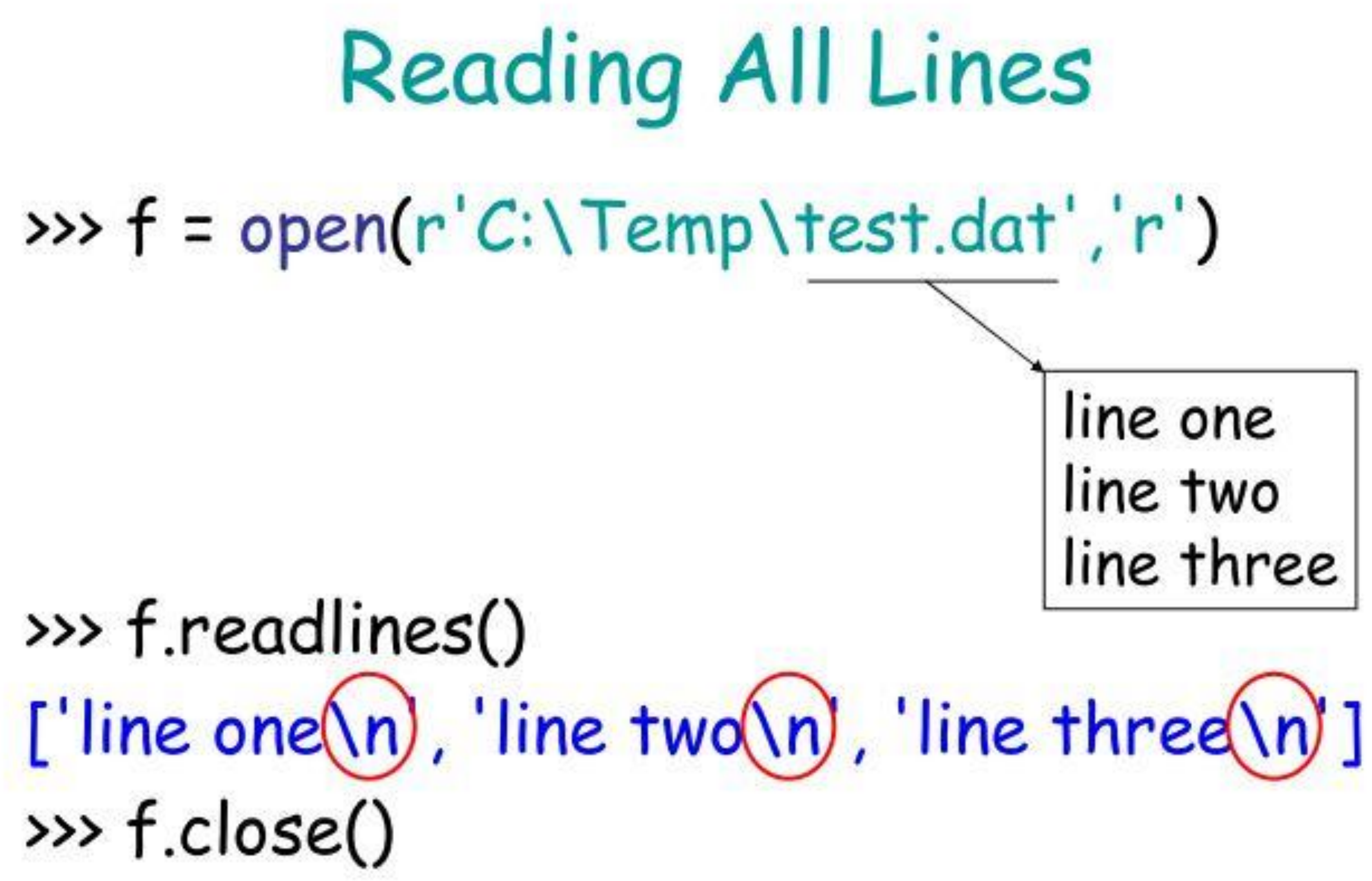
- printable chars: a…z, A…Z, 0…9, ~!@#$
- newline: '\n'
- whitespace: ' ', '\t', '\n' ...

Reading Line by Line

```
>>> f = open(r'C:\Temp\test.dat','r')
>>> f.readline()
'line one\n'
>>> f.readline()
'line two\n'
>>> f.readline()
'line three\n'
>>> f.close()
```



```
line one
line two
line three
```



Reading All Lines

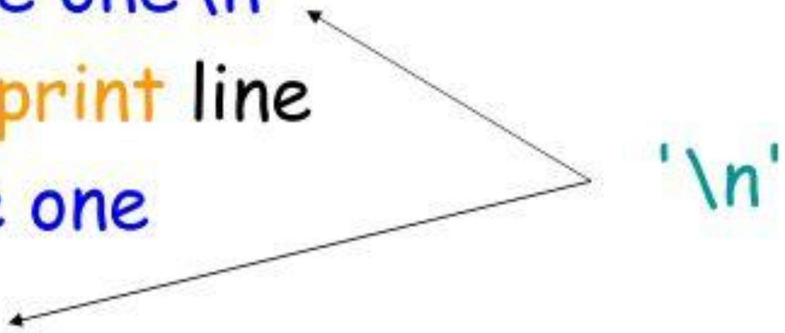
```
>>> f = open(r'C:\Temp\test.dat','r')
```

```
line one
line two
line three
```

```
>>> f.readlines()
['line one\n', 'line two\n', 'line three\n']
>>> f.close()
```

'\n' ?

```
>>> line = f.readline()
>>> line
'line one\n'
>>> print line
line one

>>>
```



'\n'

String Stripping

```
>>> help(''.strip)
strip(...)
    S.strip(...) -> string

    Return a copy of the string S with
    leading and trailing whitespace removed.
    ...
```

Strip the Newline

```
>>> line = f.readline()

>>> line
'line one\n'

>>> line.strip()
'line one'
```

Formatting

There is the format operator → %

Syntax:

"format string" % (tuple, of, expressions)

The result is a string!

For every format sequence in the string there must have an expression in the tuple.

Format sequences

The general form of a format sequence is:

'%' + <format flags> + <format code>

Format codes

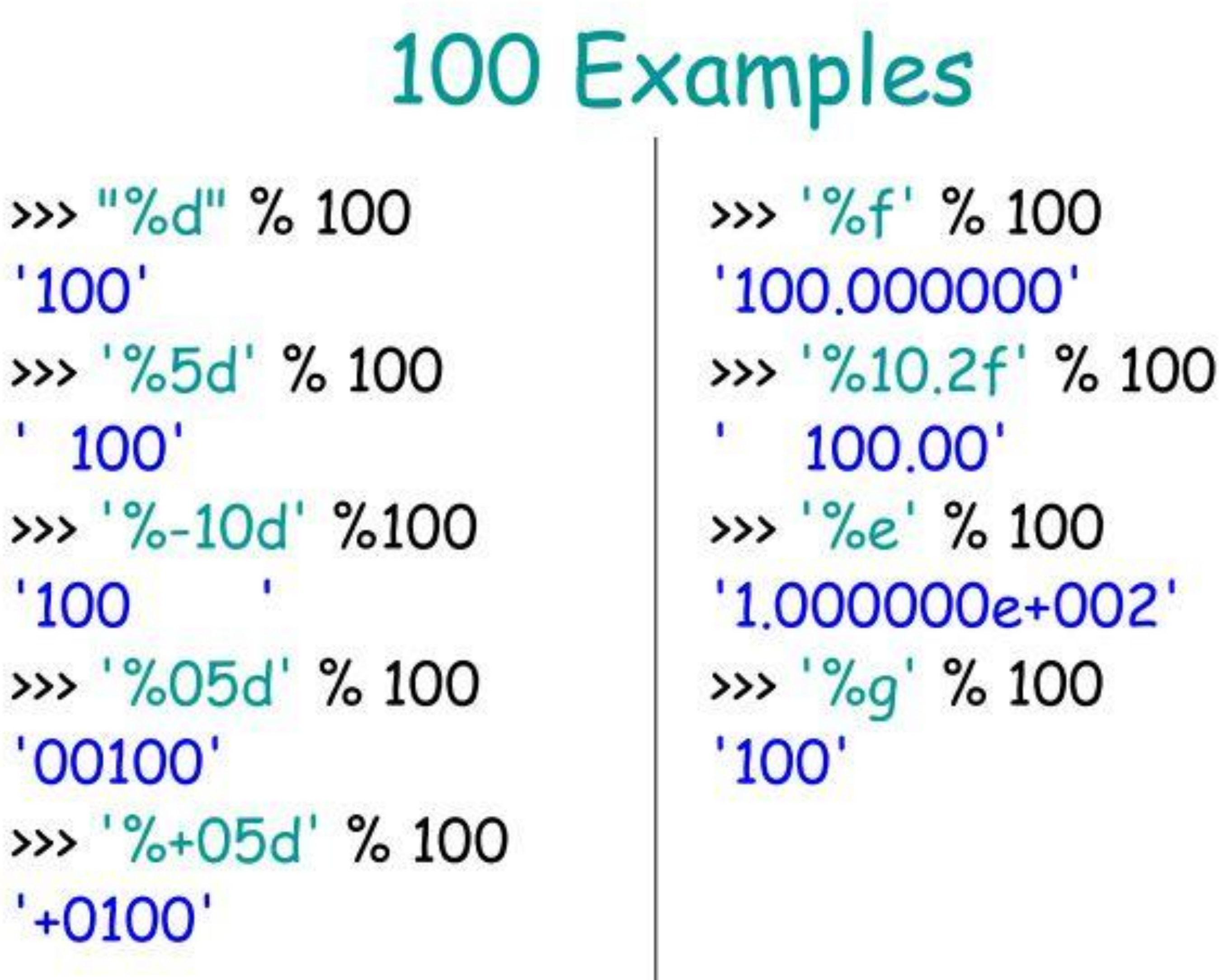
- Format sequences in the format string all start with a '%'.

Code	Prints as
%d	decimal.
%f	floating point
%e	exponential format
%g	Shortest of %d, %e or %f
%c	character
%s	string

Format flags

- Format flags further specify the formatting.

Flag	Meaning
0	Use zero padding
-	Left adjustment
space	Put space before positive
+	Use a sign: +/-
.	Width & decimal places



100 Examples

```
>>> "%d" % 100
'100'
>>> '%5d' % 100
'  100'
>>> '%-10d' %100
'100       '
>>> '%05d' % 100
'00100'
>>> '%+05d' % 100
'+0100'
```

```
>>> '%f' % 100
'100.000000'
>>> '%10.2f' % 100
'    100.00'
>>> '%e' % 100
'1.000000e+002'
>>> '%g' % 100
'100'
```

More Examples

```
>>> "%s has %d students." % ('ITC', 400)
'ITC has 400 students.'
```

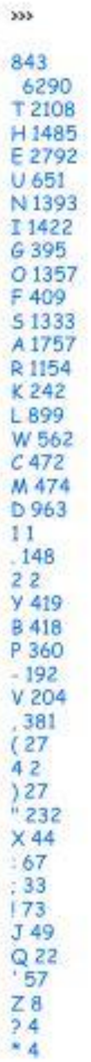
Printing a '%'?

```
>>> 'The weather improved %d%%' % 100
'The weather improved 100%'
```

Count chars using 2 lists

```
f = open('M:\\education\\Python-2007\\snark12.txt')

story = f.read().upper()

f.close()

chars = [ ]
counts = [ ]
for c in story:
    if c in chars:
        i = chars.index(c)
        counts[i] = counts[i] + 1
    else:
        chars.append(c)
        counts.append(1)
...
```

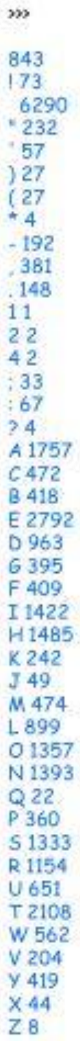
```
...

i = 0
while i < len(chars):
    print chars[i], counts[i]
    i = i + 1
```



```
>>>

843
  6290
T 2108
H 1485
E 2792
U 651
N 1393
I 1422
G 395
O 1357
F 409
S 1333
A 1757
R 1154
K 242
L 899
W 562
C 472
M 474
D 963
I 1
. 148
2 2
Y 419
B 418
P 360
- 192
V 204
, 381
( 27
4 2
) 27
" 232
X 44
: 67
; 33
! 73
J 49
Q 22
' 57
Z 8
? 4
* 4
```

Count chars using 1 dict

```
f = open('M:\\education\\Python-2007\\snark12.txt')

story = f.read().upper()

f.close()

chars = { }

for c in story:
    if c in chars:
        chars[c] = chars[c] + 1
    else:
        chars[c] = 1

for c in chars:
    print c, chars[c]
```



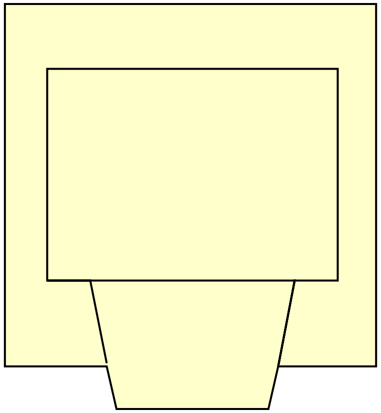
```
>>>

843
! 73
  6290
" 232
' 57
) 27
( 27
* 4
- 192
, 381
. 148
1 1
2 2
4 2
; 33
: 67
? 4
A 1757
C 472
B 418
E 2792
D 963
G 395
F 409
I 1422
H 1485
K 242
J 49
M 474
L 899
O 1357
N 1393
Q 22
P 360
S 1333
R 1154
U 651
T 2108
W 562
V 204
Y 419
X 44
Z 8
```

Word Count

```
from string import whitespace, punctuation

f = open('snark12.txt')

book = f.readlines()

f.close()

wordcount = { }

for line in book:
    line = line.split()
    for word in line:
        word = word.strip(whitespace + punctuation).lower()
        if word in wordcount:
            wordcount[word] = wordcount[word] + 1
        else:
            wordcount[word] = 1

wordcount = wordcount.items()
wordcount.sort(cmp=lambda x, y: cmp(x[1], y[1]), reverse=True)

print wordcount[0:10]
```

the	352
and	163
a	132
to	127
it	111
of	95
in	91
with	88
that	83
he	83

Time for Practical and Assignment



Date of Assignment Submission: 15.10.2013 by 1700hrs