Lesson 8: Sorting, File IO

Fundamentals of Text Processing for Linguists

Na-Rae Han

Objectives

- Sorting
- File IO: reading from and writing to a file
  - WD (working directory) and file path

Manipulate list order: `.reverse()`, `.sort()`

```
>>> li = [2, 1, 3, 4]
>>> li.reverse()
>>> li
[4, 3, 1, 2]
```

`.reverse()`
reverses a list
in place
(= in memory)

```
>>> li = [2, 1, 3, 4]
>>> li.sort()
>>> li
[1, 2, 3, 4]
>>> li.sort(reverse=True)
>>> li
[4, 3, 2, 1]
```

`.sort()`
sorts a list
in place

.sort() vs. sorted()

- list.sort()

```
>>> li = [2, 3, 1, 4]
>>> li.sort()
>>> li
[1, 2, 3, 4]
```

- sorted(list)

```
>>> li = [2, 3, 1, 4]
>>> sorted(li)
[1, 2, 3, 4]
```

What's the difference?

.sort() vs. sorted()

- list.sort()

```
>>> li = [2, 3, 1, 4]
>>> li.sort()
>>> li
[1, 2, 3, 4]
```

Returns nothing

li itself is changed in memory

- sorted(list)

```
>>> li = [2, 3, 1, 4]
>>> sorted(li)
[1, 2, 3, 4]
>>> li
[2, 3, 1, 4]
```

Creates and *returns* a new sorted list

li is not changed

Why use `sorted()`

- Once you sort a list through `.sort()`, the original order is lost forever.
- Often, you want to keep around the original list.
- This is when you use **`sorted()`**.

```
>>> li = [2, 3, 1, 4]
>>> sorted(li)
[1, 2, 3, 4]
>>> li
[2, 3, 1, 4]
>>> li2 = sorted(li)
>>> li2
[1, 2, 3, 4]
>>> li
[2, 3, 1, 4]
```

Assign a new name to the returned list

Sorting options

```
>>> li = ['x', 'ab', 'd', 'cde']

>>> sorted(li)
['ab', 'cde', 'd', 'x']

>>> sorted(li, reverse=True)
['x', 'd', 'cde', 'ab']

>>> sorted(li, key=len)
['x', 'd', 'ab', 'cde']

>>> sorted(li, key=len, reverse=True)
['cde', 'ab', 'x', 'd']
```

Reverse alphabetical order

Use `len()` function as key: Order by string length

Likewise, but in the descending order

sorted() returns a list

- `sorted()` is not a list method but a general function: it also works on other sequence types (strings & tuples) and dictionaries.
- No matter the input type, it returns a **list**.

```
>>> seasons = ('spring', 'summer', 'fall', 'winter')
>>> sorted(seasons, reverse=True)
['winter', 'summer', 'spring', 'fall']
```

takes a *tuple*,
returns a **sorted list**

```
>>> sorted('python')
['h', 'n', 'o', 'p', 't', 'y']
```

takes a *string*,
returns **a sorted list of characters**

```
>>> sim = {'Homer':36, 'Marge':36, 'Bart':10, 'Lisa':8}
>>> sorted(sim)
['Bart', 'Homer', 'Lisa', 'Marge']
```

takes a *dictionary*,
returns **a sorted list of keys**

Sorting dict keys with sorted()

```
>>> sim = {'Homer':36, 'Marge':36, 'Bart':10, 'Lisa':8}
>>> sim.keys()
['Homer', 'Lisa', 'Marge', 'Bart']

>>> for s in sim:
        print s, 'is', sim[s], 'years old.'
```

```
Homer is 36 years old.
Lisa is 8 years old.
Marge is 36 years old.
Bart is 10 years old.
```

Results are not in any particular order.

Sorting dict keys with sorted()

```
>>> sim = {'Homer':36, 'Marge':36, 'Bart':10, 'Lisa':8}
>>> sim.keys()
['Homer', 'Lisa', 'Marge', 'Bart']

>>> sorted(sim)
['Bart', 'Homer', 'Lisa', 'Marge']

>>> for s in sorted(sim):
        print s, 'is', sim[s], 'years old.'
```

sorted(dict)
returns a **sorted list of keys**

```
Bart is 10 years old.
Homer is 36 years old.
Lisa is 8 years old.
Marge is 36 years old.
```

Dictionary now prints out in an alphabetically sorted key order.

What you are sorting is the *dictionary keys* and NOT the dictionary itself.

Sorting dict keys by VALUE

```
>>> sim = {'Homer':36, 'Marge':36, 'Bart':10, 'Lisa':8}
>>> sim.keys()
['Homer', 'Lisa', 'Marge', 'Bart']

>>> sorted(sim, key=sim.get)
['Lisa', 'Bart', 'Homer', 'Marge']

>>> for s in sorted(sim, key=sim.get) :
        print s, 'is', sim[s], 'years old.'
```

dict[x] and dict.get(x) do the same thing: retrieving the **dict value**.

Returns a sorted list keys in the order of their value.

```
Lisa is 8 years old.
Bart is 10 years old.
Homer is 36 years old.
Marge is 36 years old.
```

Dictionary now prints out in the value's sorted order.

Try it out

2 minutes

```
>>> sim = {'Homer':36, 'Marge':36, 'Bart':10, 'Lisa':8,
'Maggie':1}
>>> sim.keys()
['Homer', 'Lisa', 'Maggie', 'Marge', 'Bart']

>>> sorted(sim, key=sim.get)
['Maggie', 'Lisa', 'Bart', 'Homer', 'Marge']

>>> for s in sorted(sim, key=sim.get):
        print s, 'is', sim[s], 'years old.'
```

Also try:
key=len
reverse=True

```
Maggie is 1 years old.
Lisa is 8 years old.
Bart is 10 years old.
Homer is 36 years old.
Marge is 36 years old.
```

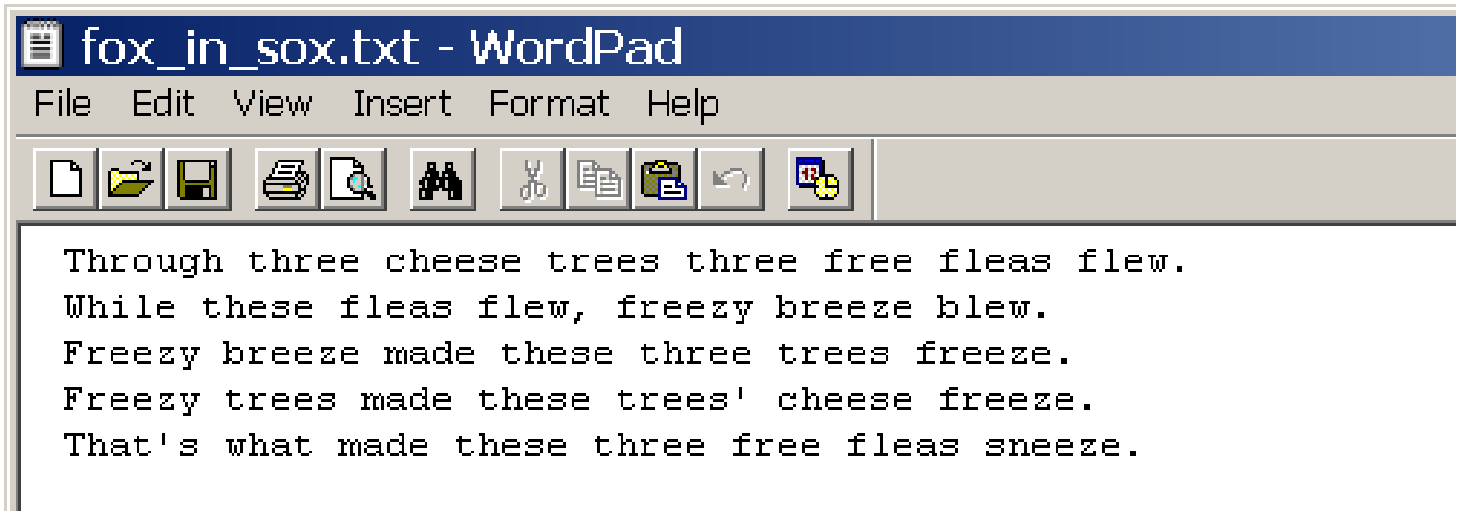
Finally, working with a file

- How to write a script that reads in a text file and processes it? How to write out the result?

- **File IO** (Input/Output)
  - Read in content of a file
  - Write out to a file
  - Write content into an existing file

- Also: "pickling" a Python data object
  - A list, a dictionary, ... → LATER CLASS

fox_in_sox.txt



- The text has 5 lines.

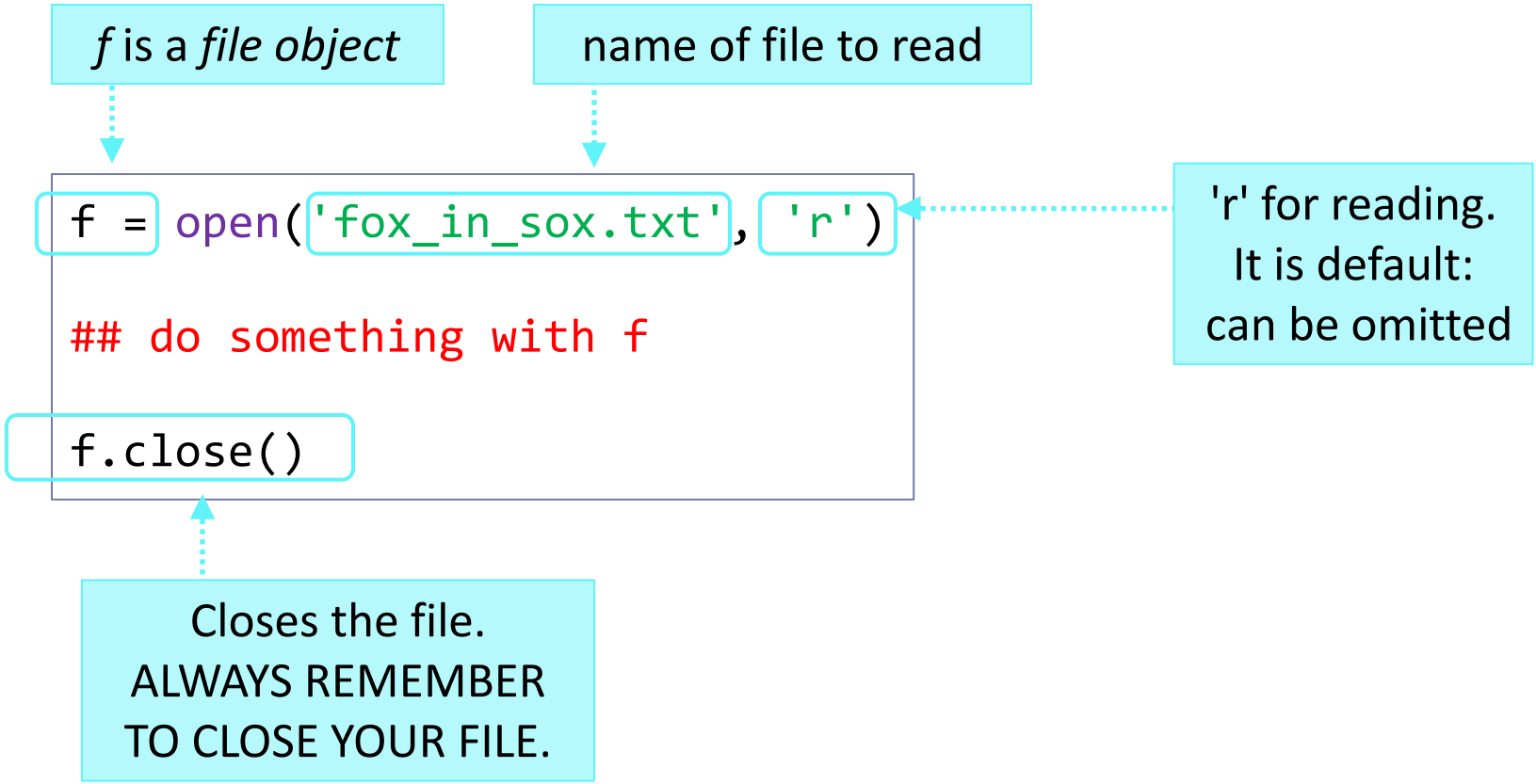
Opening a text file for reading

```
f = open('fox_in_sox.txt', 'r')

## do something with f

f.close()
```

Opening a text file for reading



(1) Reading entire text as a single string

```
f = open('fox_in_sox.txt', 'r')

text = f.read()
print text,

f.close()
```

`f.read()`
Variable *text* is a string obj whose value is the **entire text**

',' is necessary.
The text already includes line breaks, and without ',' `print` will add an extra line break

```
>>> ========================= RESTART ========
>>>
Through three cheese trees three free fleas flew.
While these fleas flew, freezy breeze blew.
Freezy breeze made these three trees freeze.
Freezy trees made these trees' cheese freeze.
That's what made these three free fleas sneeze.
>>>
```

(2) Reading entire text as a list of lines

```
f = open('fox_in_sox.txt')

lines = f.readlines()
print lines

f.close()
```

`f.readlines()` reads in the text as a **list of lines**.

lines is a **list of strings**, where each string is a single line.

```
>>> ======================== RESTART ======================
>>>
['Through three cheese trees three free fleas flew.\n', 'While
these fleas flew, freezy breeze blew.\n', 'Freezy breeze made
these three trees freeze.\n', "Freezy trees made these trees'
cheese freeze.\n", "That's what made these three free fleas
sneeze.\n"]
>>>
```

(3) Reading text in, line by line

```
f = open('fox_in_sox.txt')

for line in f :
    print line,

f.close()
```

`for x in fileobj` routine reads in file, line by line

Again, don't forget ','

- This for loop routine is more memory efficient
- File is processed line by line
- No new Python object (string, list, etc.) is created that holds the entire content of the file!
- Recommended for processing large files

Entire file content can be read in only once per opening

```
f = open('fox_in_sox.txt', 'r')

text = f.read()
lines = f.readlines()

print len(text)
print len(lines)

f.close()
```

Reads in the entire file content

Has nothing more to read!

Prints "0"

- File objects have a concept of a "cursor".
- After the entire content is read in, the cursor moves to the end of the file.
- At that point, the next attempt to read from the file will return nothing.

Try it out

3 minutes

- Download this file where you usually save your script:
  http://www.pitt.edu/~naraehan/ling1991/fox_in_sox.txt
- And try these scripts:

```
f = open('fox_in_sox.txt', 'r')

for line in f :
    print line,

f.close()
```

substitute

```
lines = f.readlines()
print lines
for l in lines :
    if 'tree' in l : print l,
```

```
text = f.read()
print text,
print text.split()
```

Try it out

2 minutes

- Download this file where you usually save your script:
  http://www.pitt.edu/~naraehan/ling1991/stopwords-english.txt
- And in your shell, open the file and turn the words into a list.

```
>>> f = open('stopwords-english.txt', 'r')
>>> stopwords = []
                    ??
>>> f.close()
>>> print stopwords
['i', 'me', 'my', 'myself', 'we', 'our',
'ourselves', 'you', 'your', 'yours',
'yourself', 'yourselves', 'he', 'him', ...
```

```
…
but
if
or
because
…
```

Try it out

2 minutes

- Download this file where you usually save your script:
  http://www.pitt.edu/~naraehan/ling1991/stopwords-english.txt
- And in your shell, open the file and turn the words into a list.

```
>>> f = open('stopwords-english.txt', 'r')
>>> stopwords = []
>>> for line in f :
        stopwords.append(line.strip())

>>> f.close()
>>> print stopwords
['i', 'me', 'my', 'myself', 'we', 'our',
'ourselves', 'you', 'your', 'yours',
'yourself', 'yourselves', 'he', 'him', ...
```

Removes line break in `'but\n'`

```
…
but
if
or
because
…
```

Try it out

2 minutes

- Download this file where you usually save your script:
  http://www.pitt.edu/~naraehan/ling1991/stopwords-english.txt
- And in your shell, open the file and turn the words into a list.

```
>>> f = open('stopwords-english.txt', 'r')
>>> text = f.read()
>>> stopwords = text.split()
>>> f.close()
>>> print stopwords
['i', 'me', 'my', 'myself', 'we', 'our',
'ourselves', 'you', 'your', 'yours',
'yourself', 'yourselves', 'he', 'him', ...
```

Also works.

```
…
but
if
or
because
…
```

Writing to a file

```
f = open('foo.txt', 'w')

f.write('Roses are red,\n')
f.write('violets are blue.\n')

f.close()
```

'w' option
Opens file for *writing*

`f.write(str)`
Writes a string to file f

- Creates file foo.txt,

and writes two lines →

foo.txt

```
Roses are red,
violets are blue.
```

Writing works one string at a time

```
f = open('foo.txt', 'w')

f.write('Roses are red,\n', 'violets are blue.\n')

f.close()
```

```
>>> ========================== RESTART ======================
>>>
Traceback (most recent call last):
  File "F:\foo.py", line 3, in <module>
    f.write('Roses are red,\n', 'violets are blue.\n')
TypeError: function takes exactly 1 argument (2 given)
>>>
```

Line breaks must be supplied

```
f = open('foo.txt', 'w')

f.write('Roses are red,')
f.write('violets are blue.')

f.close()
```

Without **\n**, these will be printed on a single line.

foo.txt

```
Roses are red,violets are blue.
```

- `print` by default adds a line break. (',' at the end suppresses it.)
- The `.write()` file method DOES NOT add a line break.
- Line breaks '\n' must be explicitly supplied.

Can only write string types

```
f = open('foo.txt', 'w')

pi = 3.14159265
f.write('The value of pi is:\n')
f.write(pi)

f.close()
```

pi is float type.
`.write()` can only take a string argument.

```
>>> ========== RESTART ==========
>>>
Traceback (most recent call last):
  File "F:\foo.py", line 5, in <module>
    f.write(pi)
TypeError: expected a character buffer object
>>>
```

Can only write string types

```
f = open('foo.txt', 'w')

pi = 3.14159265
f.write('The value of pi is:\n')
f.write(str(pi))

f.close()
```

`str()` turns *pi* into a **string**. Works now.

foo.txt

```
The value of pi is:
3.14159265
```

- `.write()` only takes a **string** as an argument.
- Any other data types (integer, float, etc.) must be first converted into string using `str()` function.

Don't forget to close file

```
f = open('foo.txt', 'w')

f.write('Roses are red\n,')
f.write('violets are blue\n.')
```

Forgot to specify `f.close()`

foo.txt

Always remember to close your file.

- File writing happens through *buffers*.
- Writing out to a file actually happens when your writing buffer is full or the file is closed.
- So, if you forget to close your file, you might find your output file to be either **empty** or **halfway written**.

Practice

2 minutes

- Try this script:

```
f = open('foo.txt', 'w')

pi = 3.14159265
f.write('The value of pi is:\n')
f.write(str(pi)+'\n')

f.close()

print 'foo.txt has been written out.'
```

Practice

3 minutes

- Finish the script so it produces the output file.

```
chom = 'Colorless green ideas sleep furiously.'
f = open('foo.txt', 'w')

                    ??

f.close()
```

foo.txt

```
Colorless is 9 characters long.
green is 5 characters long.
ideas is 5 characters long.
sleep is 5 characters long.
furiously. is 10 characters long.
```

Practice

3 minutes

- Finish the script so it produces the output file.

```
chom = 'Colorless green ideas sleep furiously.'
f = open('foo.txt', 'w')

for c in chom.split() :
    f.write(c+' is '+str(len(c))+' characters long.\n')

f.close()
```

foo.txt

```
Colorless is 9 characters long.
green is 5 characters long.
ideas is 5 characters long.
sleep is 5 characters long.
furiously. is 10 characters long.
```

Beware overwriting; file IO is costly

- Could we have done this instead?

```
chom = 'Colorless green ideas sleep furiously.'

for c in chom.split() :
    f = open('foo.txt', 'w')
    f.write(c+' is '+str(len(c))+' characters long.\n')
    f.close()
```

Problem #2:
Opening and closing files is resource-intensive.
This code is INEFFICIENT.

foo.txt

```
furiously. is 10 characters long.
```

Problem #1:
File gets over-written with each iteration!

File location and path: WD

- When a file is referred to by its name only, it is assumed to be in the current **working directory** (**WD**).

- In *scripts*, WD is the directory where your **script** is.

```
f = open('fox_in_sox.txt')
# read file
f.close()

outf = open('results.txt', 'w')
# write out to file
outf.close()
```

File to read must be in the same directory as the script

File will be created in the directory where the script is

WD in Python shell

- In IDLE *shell* or in *command-line*, WD may be initially set to:
  - Where your python.exe file is (Windows standard installation):

    C:\Python27
  - The directory where you invoked python (OS-X, Linux, Unix) :

    /Users/naraehan/Documents

```
>>> f = open('fox_in_sox.txt')
Traceback (most recent call last):
  File "<pyshell#48>", line 1, in <module>
    f = open('fox_in_sox.txt')
IOError: [Errno 2] No such file or directory: 'fox_in_sox.txt'
```

Error:
The file is not in your shell's current working directory.

Discovering and changing your WD

```
>>> import os
>>> os.getcwd()
'D:\\Lab'
```

os module must be imported first.
`os.getcwd()` displays "current WD"

```
>>> os.chdir('C:\\Users\\narae\\Documents')

>>> os.getcwd()
'C:\\Users\\narae\\Documents'
```

`os.chdir()` changes current WD.
It is now set to
`C:\Users\narae\Document`
(Windows)

In OS X, directories look like
`/Users/naraehan/Document`

Wrap-up

- **Next class**
  - Working with modules
  - Text processing

- **Exercise6**
  - http://www.pitt.edu/~naraehan/ling1901/exercise.html#ex6
  - Due Tuesday midnight