Processing Data from Files

- So far:
  - **Inputs**:
    - … from user
    - … "hard-wired" into program
  - **Outputs**:
    - … "printing" on the screen
- In practice, usually:
  - **Input from file**
  - **Output to file**

File Processing

- The process of **opening** a file involves associating a file on disk with an object in memory.
- We can manipulate the file by manipulating this object.
  - **Read from** the file
  - **Write to** the file

File Processing

- When done with the file, it needs to be **closed**. Closing the file causes any outstanding operations and other bookkeeping for the file to be completed.
- In some cases, not properly closing a file could result in data loss.

File Processing

- **Reading** a file into a word processor
  - File opened
  - Contents read into RAM
  - File closed
  - Changes to the file are made to the copy stored in memory, not on the disk.
    - Aside: who uses Dropbox?

    ➔ interesting issues with access control (easy to "shoot yourself in the foot", when multiple users edit same file)

File Processing

- **Saving** a word processing file
  - The original file on the disk is reopened in a mode that will allow writing (this actually erases the old contents)
  - File writing operations copy the version of the document in memory to the disk
  - The file is closed

File Processing

- Working with **text files** in Python
  - Associate a disk file with a file object using the open function
    access through this var!
    '~/foo/bar/test.tx'
    'r' ~ read, 'w' ~ write, 'a' ~ append
    <filevar> = open(<name>, <mode>)
  - Name is a string with the actual file name on the disk. The mode is either '**r**' or '**w**' depending on whether we are reading or writing the file.
  - MyInfile = open("numbers.dat", "r")

MyInputFile

n

variable.method(...)
object.method(...)

File Methods

- <file>.**read**() – returns the **entire** remaining contents of the file as a single (possibly large, multi-line) string
- <file>.**readline**() – returns the **next line** of the file. This is all text up to *and including* the next newline character
- <file>.**readlines**() – returns a **list** of the remaining lines in the file. Each list item is a single line including the newline characters.

File Processing

- Another way to loop through the contents of a file is to read it in with readlines and then loop through the resulting list:
- MyInfile = open(someFile, "r")
  for line in (MyInfile.readlines)():
          # Line processing here
  MyInfile.close()

```
L = MyInfile.readlines()
for line in L:
    ...
```

File Processing

- Python treats the file itself as a sequence of lines
  Very convenient!

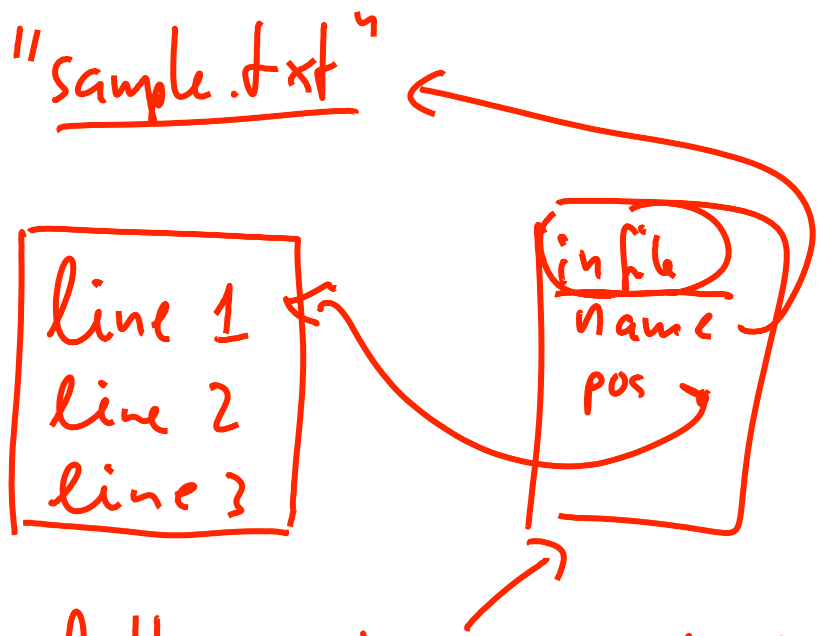
-
  ```
  MyInfile = open(someFile, "r")
  for line in MyInfile:
      # process the line here
  MyInfile.close()
  ```
- Bottom line:
  - Processing a text file, line by line?? Use a **for** loop!

File Processing

- Opening a file for **writing** prepares the file to receive data
- If you open an existing file for writing, **you wipe out the file's old contents**. If the named file does not exist, **a new one is created**.
- Outfile = open("mydata.out", "**w**")
- print(<expressions>, file=Outfile)
  - Alternative (and main option in Python 2):
    - Outfile.write(…)

      .writelines(L)
- This is very convenient (better than in Python 2!):
  1. Develop code with print(..) statements
  2. When all works, add "file = Outfile" (and all what goes with it) to the program!

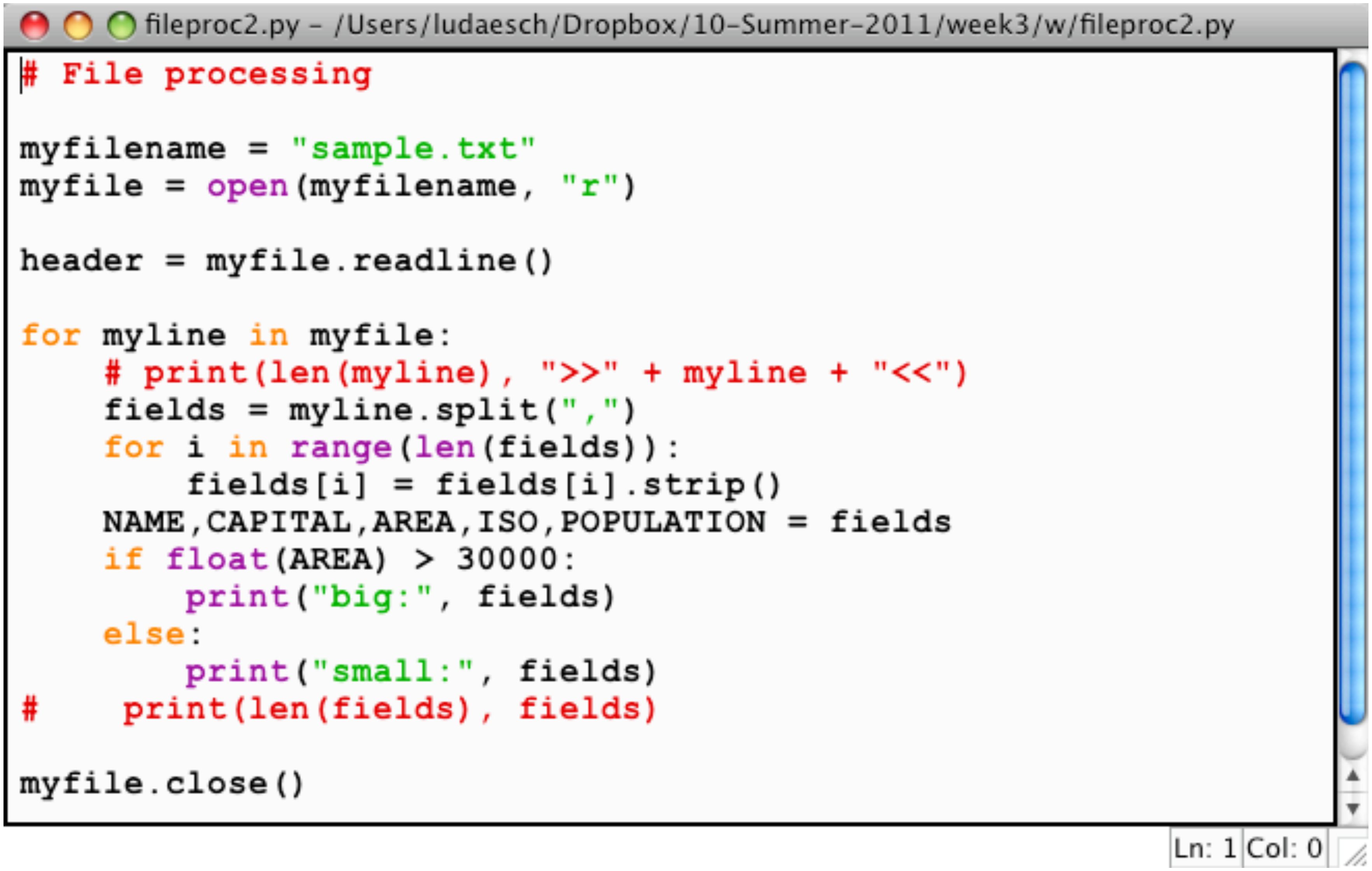
File Processing



```
fname = "sample.txt"
infile = open(fname, "r")
   :
   infile.readline()
   :
infile.close()
```

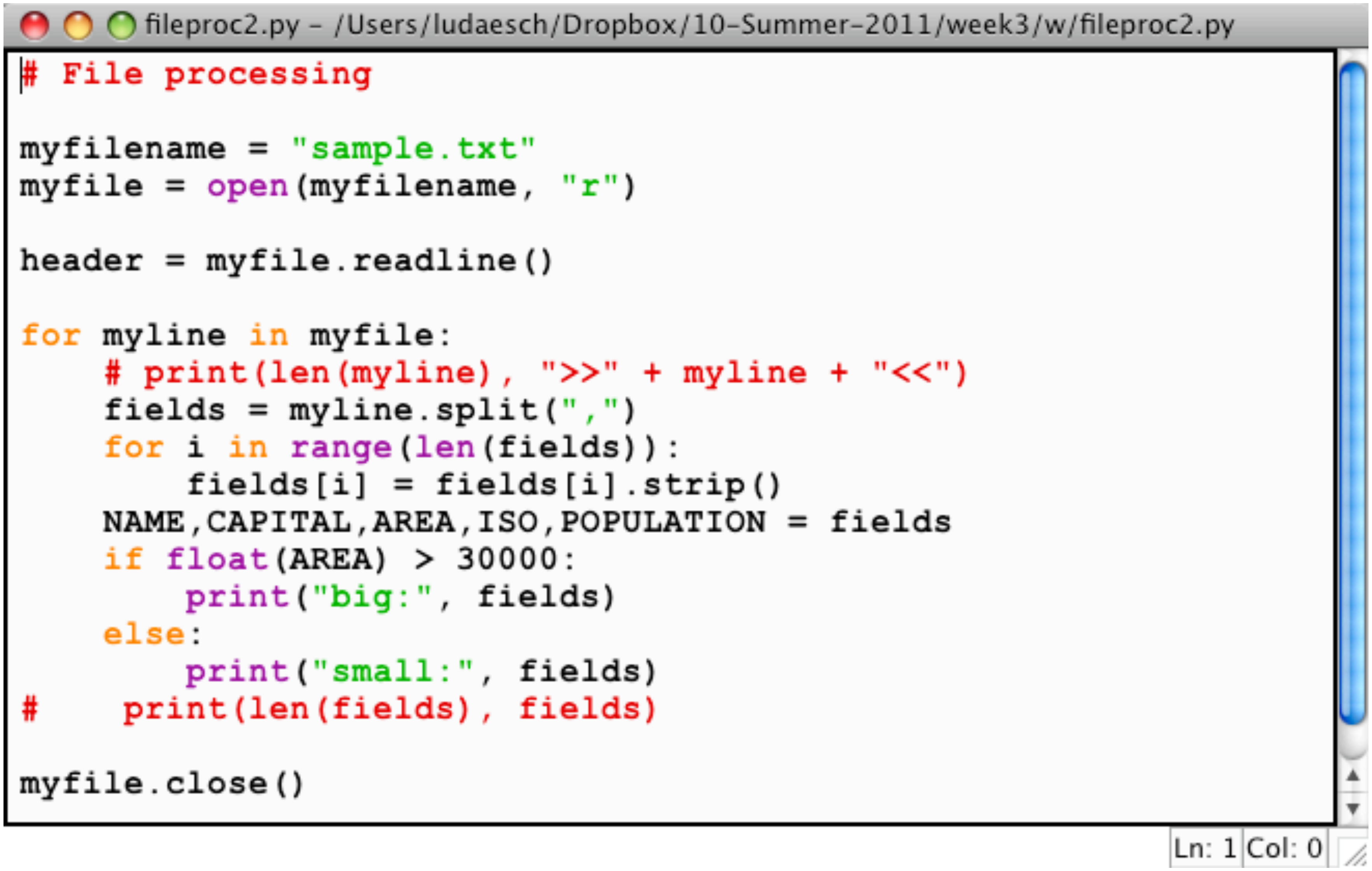
Python's internal datastructure
to remember filename, current position in file, etc.

File Processing: Examples


fileproc2.py - /Users/ludaesch/Dropbox/10-Summer-2011/week3/w/fileproc2.py

```
# File processing

myfilename = "sample.txt"
myfile = open(myfilename, "r")

header = myfile.readline()

for myline in myfile:
    # print(len(myline), ">>" + myline + "<<")
    fields = myline.split(",")
    for i in range(len(fields)):
        fields[i] = fields[i].strip()
    NAME,CAPITAL,AREA,ISO,POPULATION = fields
    if float(AREA) > 30000:
        print("big:", fields)
    else:
        print("small:", fields)
#       print(len(fields), fields)

myfile.close()
```

From Reading to Writing Files …


fileproc2.py – /Users/ludaesch/Dropbox/10-Summer-2011/week3/w/fileproc2.py

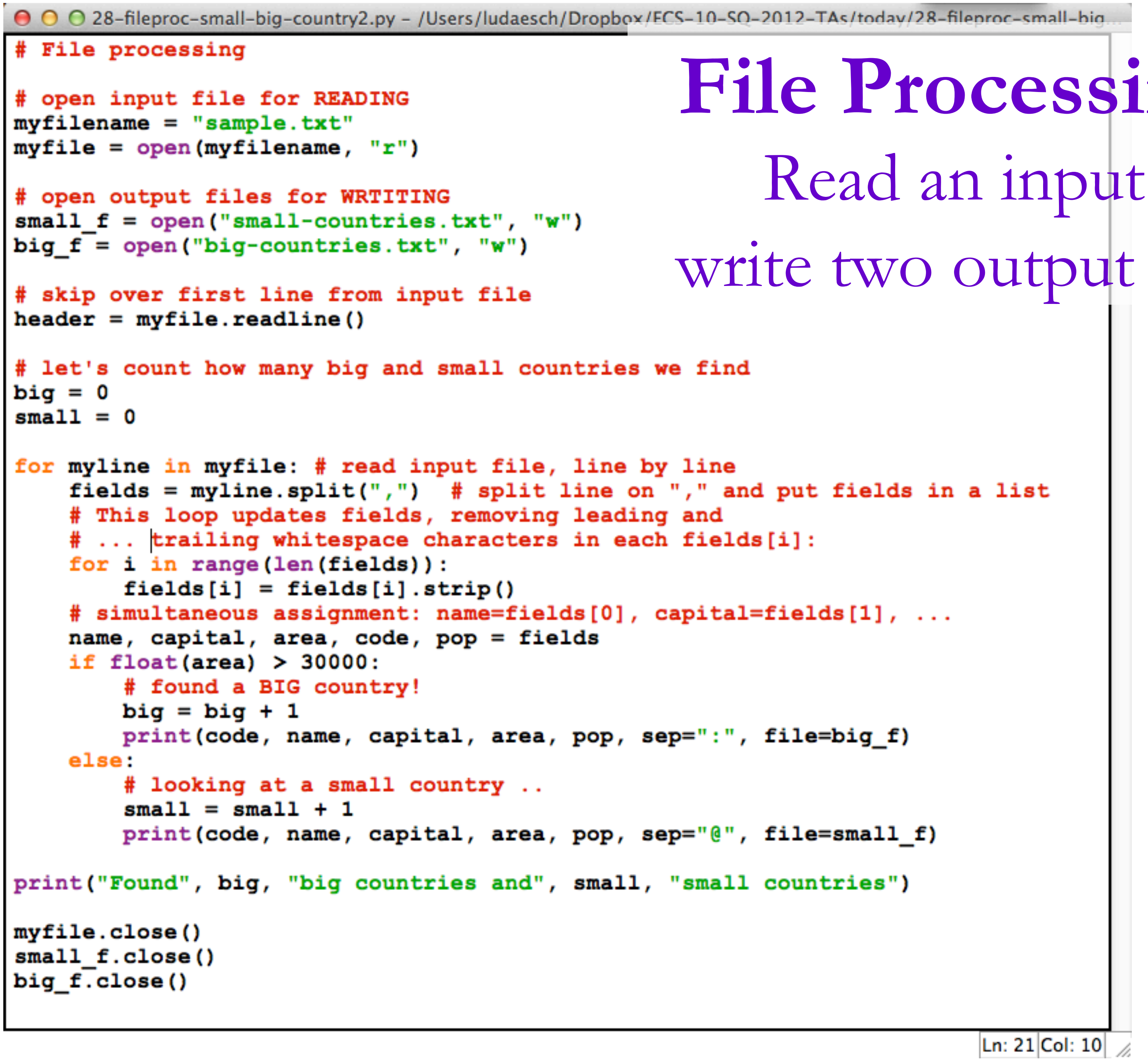
```python
# File processing

myfilename = "sample.txt"
myfile = open(myfilename, "r")

header = myfile.readline()

for myline in myfile:
    # print(len(myline), ">>" + myline + "<<")
    fields = myline.split(",")
    for i in range(len(fields)):
        fields[i] = fields[i].strip()
    NAME,CAPITAL,AREA,ISO,POPULATION = fields
    if float(AREA) > 30000:
        print("big:", fields)
    else:
        print("small:", fields)
#     print(len(fields), fields)

myfile.close()
```

Ln: 1 Col: 0

File Processing:

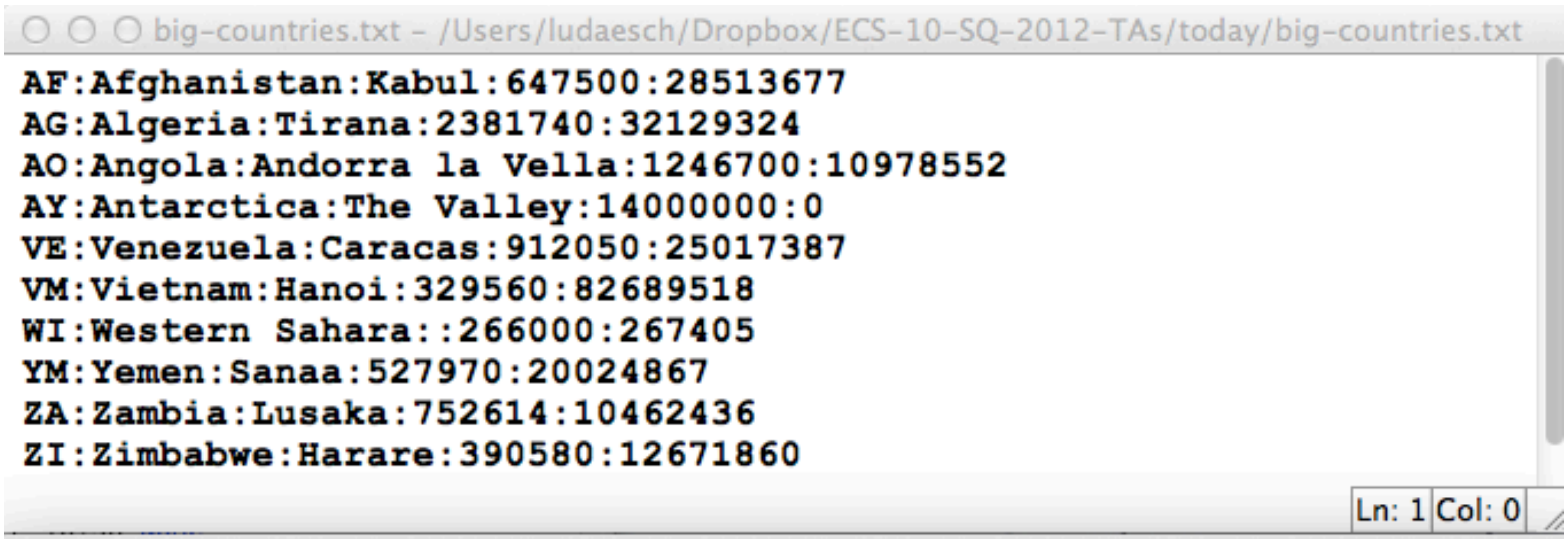
Read an input file,

write two output files


28-fileproc-small-big-country2.py - /Users/ludaesch/Dropbox/ECS-10-SQ-2012-TAs/today/28-fileproc-small-big...

```python
# File processing

# open input file for READING
myfilename = "sample.txt"
myfile = open(myfilename, "r")

# open output files for WRTITING
small_f = open("small-countries.txt", "w")
big_f = open("big-countries.txt", "w")

# skip over first line from input file
header = myfile.readline()

# let's count how many big and small countries we find
big = 0
small = 0

for myline in myfile: # read input file, line by line
    fields = myline.split(",")  # split line on "," and put fields in a list
    # This loop updates fields, removing leading and
    # ... trailing whitespace characters in each fields[i]:
    for i in range(len(fields)):
        fields[i] = fields[i].strip()
    # simultaneous assignment: name=fields[0], capital=fields[1], ...
    name, capital, area, code, pop = fields
    if float(area) > 30000:
        # found a BIG country!
        big = big + 1
        print(code, name, capital, area, pop, sep=":", file=big_f)
    else:
        # looking at a small country ..
        small = small + 1
        print(code, name, capital, area, pop, sep="@", file=small_f)

print("Found", big, "big countries and", small, "small countries")

myfile.close()
small_f.close()
big_f.close()
```

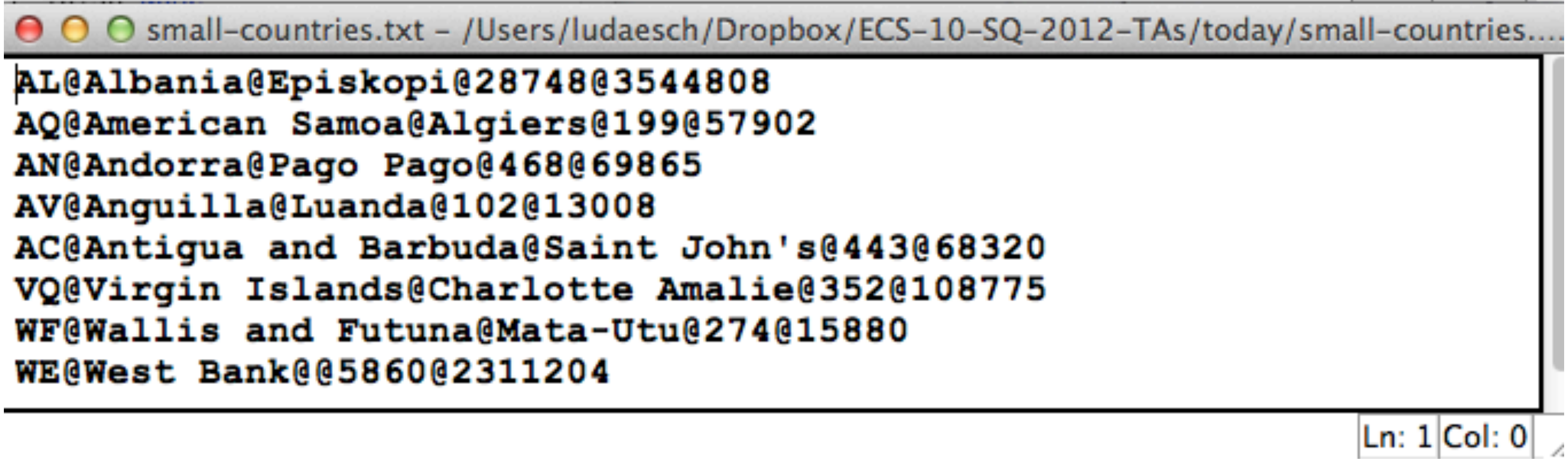
… two files are generated:



big-countries.txt – /Users/ludaesch/Dropbox/ECS-10-SQ-2012-TAs/today/big-countries.txt

```
AF:Afghanistan:Kabul:647500:28513677
AG:Algeria:Tirana:2381740:32129324
AO:Angola:Andorra la Vella:1246700:10978552
AY:Antarctica:The Valley:14000000:0
VE:Venezuela:Caracas:912050:25017387
VM:Vietnam:Hanoi:329560:82689518
WI:Western Sahara::266000:267405
YM:Yemen:Sanaa:527970:20024867
ZA:Zambia:Lusaka:752614:10462436
ZI:Zimbabwe:Harare:390580:12671860
```

Ln: 1 Col: 0



small-countries.txt – /Users/ludaesch/Dropbox/ECS-10-SQ-2012-TAs/today/small-countries....

```
AL@Albania@Episkopi@28748@3544808
AQ@American Samoa@Algiers@199@57902
AN@Andorra@Pago Pago@468@69865
AV@Anguilla@Luanda@102@13008
AC@Antigua and Barbuda@Saint John's@443@68320
VQ@Virgin Islands@Charlotte Amalie@352@108775
WF@Wallis and Futuna@Mata-Utu@274@15880
WE@West Bank@@5860@2311204
```

Ln: 1 Col: 0