Week 6: File processing

Textbook, Chapter 8.

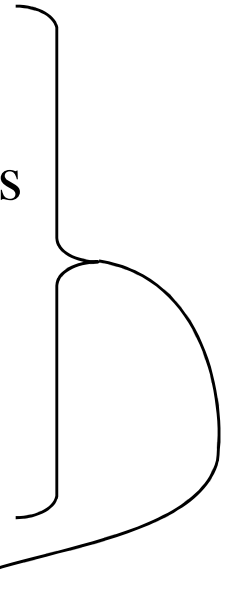
Types of file-processing tasks

Our Focus

- HUGE range, e.g.
  - Numerical /scientific data processing (e.g. rainfall data)
  - Commercial data processing (e.g. files of account transactions)
  - Document processing (e.g. MS Word documents)
  - Programming language compilation (e.g. a Fortran program)
  - Image processing (e.g. green screening)
  - Internet data harvesting (e.g. web-crawling for email addresses)
  - ...

Steps in processing numerical data

1. Open the file
2. Extract the data from the file
   - May be as simple as splitting each line in a *.csv* file or as complex as parsing an XML file
3. Process the data
4. Output/display the results



May have to interleave these steps for large files
(can't fit all data in memory)

Opening files

- We've already seen the usual opening of a local file:
  - data_file = open("data/resources/blah.txt", "r")

See slides 112, 113

    - File "name" is most generally a *file path*, i.e. a path to the file within the directory tree
- But we can also open Internet resources as files, e.g.:

```
import urllib  # The URL (Uniform Resource Locator) library
url = "http://www.cosc.canterbury.ac.nz/open/teaching/"
web_page = urllib.urlopen(url)
for line in web_page:
    print line,
web_page.close()  # Should always do this – earlier code was lazy ☹
```

What open *returns*

- Value returned by *open(filename, 'r')* is an object that behaves like a book (representing the contents of the file) and a bookmark
  - Bookmark is initially at start of book
  - When you *read* the book, you starting reading from the bookmark
  - The bookmark is left just after the last char/word/line you read
- Similarly for *open(filename, 'w')* except now the book is like an initially empty exercise book, and you always start writing at the bookmark location.
  - Bookmark updates after each write

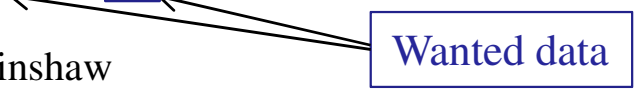
Extracting data

- Data files given to you so far have been "sanitised"
- Real data files usually have lots of extraneous info
  - Headers, footers, irrelevant data, etc
  - For example, see next slide
    - The result of querying for sunshine data at Christchurch from http://cliflo.niwa.co.nz
- Need an *algorithm* to extract just the required data
  - e.g. month, day, sunshine from following slide

cliflo.niwa.co.nz query result (csv)

Station information:
Name,Agent Number,Network Number,Latitude (dec.deg),Longitude (dec.deg),Height (m),...
Christchurch Aero,4843,H32451,-43.493,172.537,37,G,N/A
Note: Position precision types are: "W" = based on whole minutes, "T" = estimated to tenth minute,
"G" = derived from gridref , "E" = error cases derived from gridref,
"H" = based on GPS readings (NZGD49), "D" = by definition i.e. grid points.

Sunshine: Daily
Station,Date(NZST),Time(NZST),Amount(Hrs),Period(Hrs),Freq
Christchurch Aero,20100101,2259,9.9,24,D
Christchurch Aero,20100102,2259,7.1,24,D
Christchurch Aero,20100103,2259,1.8,24,D
Christchurch Aero,20100104,2259,9.7,24,D
...
Christchurch Aero,20100320,2259,1.8,24,D
Christchurch Aero,20100321,2259,0.3,24,D
Christchurch Aero,20100322,2259,5.3,24,D
Christchurch Aero,20100323,2259,9.6,24,D
Christchurch Aero,20100324,2259,1.0,24,D



UserName is = angusmcgurkinshaw
Total number of rows output = 83
Number of rows remaining in subscription = 1999917
Copyright NIWA 2010 Subject to NIWA's Terms and Conditions
See: http://cliflo.niwa.co.nz/pls/niwp/doc/terms.html
Comments to: cliflo@niwa.co.nz

Algorithm #1 for extracting data

- Many possibilities. One is:

skip lines until we get an empty line
skip two more lines

} More robust against changes in file format than "skip 9 lines"

```
read a line
while line not empty:  # A blank line terminates actual data rows
    split line into pieces separated by comma
    date = piece[1]
    get month and day from date
    sunshine = float(piece[3])
    process month, day, sunshine data point (e.g. write to another file)
    read a line
```

"Idiom 1" from last chapter

Question: what happens if the data file doesn't contain the expected two blank lines?

Code

Direct translation from pseudocode

```
infile = open("sunshine.txt")
line = infile.readline()
while line != '\n':
    line = infile.readline()

infile.readline()
infile.readline()

line = infile.readline()
while line != '\n':
    pieces = line.split(',')
    date = pieces[1]
    month = int(date[4:6])
    day = int(date[6:8])
    sunshine = float(pieces[3])
    print month, day, sunshine
    line = infile.readline()

infile.close()
```

Variant using a function for data line processing

```
def process_data_line(line):
    pieces = line.split(',')
    date = pieces[1]
    month = int(date[4:6])
    day = int(date[6:8])
    sunshine = float(pieces[3])
    print month, day, sunshine

infile = open("sunshine.txt")
line = infile.readline()
while line != '\n':
    line = infile.readline()

infile.readline()
infile.readline()

line = infile.readline()
while line != '\n':
    process_data_line(line)
    line = infile.readline()

infile.close()
```

Algorithm #2 for extracting data

- Another is:

```
get a list of all lines in file
make a list of all those lines (after line 3) beginning "Christchurch aero"
 for each of those lines:
    split line into pieces separated by comma
    date = piece[1]
    get month and day from date
    sunshine = float(piece[3])
    process month, day, sunshine data point (e.g. write to another file)
```

Simpler (?) but only works for this one base station.
Also, can't handle huge files.

Algorithm #2b

- An improvement (in terms of code reusability) is to get the station name from line 3

```
get a list of all lines in file
station_name = start of line 3, up until ","
make a list of all lines (after line 3) beginning with station_name
 for each of those lines:
    split line into pieces separated by comma
    date = piece[1]
    get month and day from date
    sunshine = float(piece[3])
    process month, day, sunshine data point (e.g. write to another file)
```

OK for any base station.
Still can't handle huge files.

Code

```
infile = open("sunshine.txt")
lines = infile.readlines()
infile.close()
station_name = lines[2].split(",")[0]  # Not lines[3] – remember 0 origin!

data = []
for line in lines[3:]:
    if line.startswith(station_name):
        data.append(line)

for line in data:
    pieces = line.split(",")
    date = pieces[1]
    month = int(date[4:6])
    day = int(date[6:8])
    sunshine = float(pieces[3])
    print month, day, sunshine
```

Which algorithm?

- Those are just 2 algorithms.
  - How many more can you find?
- Which is better?
  - Actually they're both pretty bad!
  - They both fail if the file format is significantly changed
  - The problem is that we've *inferred* the data format from the data
    - We really need a *specification* of the data format from the supplier
    - Acts as a *contract* ensuring (hopefully) our program continues to work in the future

Outputing the results

- Many possibilities, e.g.
  - Display textual output with *print*
  - Write a new file, e.g.
    - Pure text (e.g. csv)
    - Markup language output (e.g. HTML)
  - Graphical output
    - Maybe in a GUI
    - Maybe with *matplotlib*
      - Installed on Linux in labs but not on Windows.
    - ...

Writing output files

- Open file for writing, prepare data, e.g.

```
out_file = open('myoutput.txt', 'w')
data =  '{0},{1},{2:.3f}'.format(month, day, sunshine)
```

- Write data to file
  - Can use *print chevron* technique, e.g.

```
print >>out_file, data
```

Obsolescent: not in Python 3

  - Or directly output byte stream (usually a string), e.g.

```
out_file.write(data)
```

    - NB: must explicitly include newline character when using *write* method

- Close file

```
out_file.close()
```

Getting graphical output

- Outside official curriculum, except for GUI section later
- BUT ... scientists and engineers should at least be aware of *matplotlib*
  - See *matplotlib.sourceforge.net*
  - A plotting package modelled on the one in *matlab*
  - Multiplatform
  - Publication-quality output
  - Extremely flexible
    - But using this flexibility isn't trivial!
  - Installed only under Linux on lab machines (?)

Example program

```
import matplotlib.pyplot as plt
from datetime import date

def get_date(line):
    '''Extract the date info from the given line
     and return a Date object'''

    pieces = line.split(",")
    date_string = pieces[1]
    year = int(date_string[0:4])
    month = int(date_string[4:6])
    day = int(date_string[6:8])
    return date(year, month, day)

def get_sunshine(line):
    '''Return the float sunshine value from the
     given line'''

    return float(line.split(',')[3])
```

```
infile = open('sunshine.txt')
lines = infile.readlines()
infile.close()

station_name = lines[2].split(', ')[0]
sunshine_data = []
dates = []
for line in lines[3:]:  # Skip first 3 lines
    if line.startswith(station_name):
        sunshine = get_sunshine(line)
        sunshine_data.append(sunshine)
        dates.append(get_date(line))

plt.plot(sunshine_data, linestyle='dotted',
     marker='x')
n = len(sunshine_data)
tick_positions = range(0, n, 10)
tick_labels = []
for i in range(len(tick_positions)):
     tick_labels.append(dates[i].strftime('%d %b'))
plt.xticks(tick_positions, tick_labels, rotation=90)
plt.ylabel('Sunshine (hrs)')
plt.title('Sunshine hours, Christchurch aero, 2010')
plt.show()
```

Program's output