File Processing

Using file input and outputs

D.S. Hwang

Department of Software Science,
Dankook University

Everything should be made as simple as possible, but not simpler.

Albert Einstein, 1879-1955

Outline

Overview

Structure of data

- Understanding scientific data
- Record vs. field
- Structure of data in various format
- Python module for input and output

One Record per Line

```
Coloured fox fur production, HOPEDALE, Labrador, 1834-1842
#Source: C. Elton (1942) "Voles, Mice and Lemmings", Oxford Univ. Press
#Table 17, p.265--266
      22
      29
       2
      16
      12
      35
       8
      83
     166
```

One Record per Line

`hopedale.txt` taken form the Time Series Data Library

- No of colored fox fur pelts produced in the years 1834 −1842
- Description of the data
- Comments about the data with
- Single data on a single line.

One Record per Line

Reading a file

- No of colored fox fur pelts produced in the years 1834 −1842
- Description of the data
- Comments about the data with
- Single data on a single line.

One Record per Line

```
1 input_file = open("hopedale.txt", "r")
2 for line in input_file:
3     line = line.strip()
4     print(line)
5 input_file.close()
```

One Record per Line

The filename is hard-coded

- The name of a particular file is stored in the program.
- The program can't be used to process any other files.
- Use command−Line Arguments

```
1  import sys
2  def process_file(filename):
3      '''Open, read, and print a file.'''
4      input_file = open(filename, "r")
5      for line in input_file:
6          line = line.strip()
7          print(line)
8      input_file.close()
9  if __name__ == "__main__":
10     process_file(sys.argv[1])
```

python fn.py fname

One Record per Line

Files Over the Internet

- The module `urllib` contains a function called `urlopen` that opens a web page for reading and returns a file−like object

```
1 import urllib
2 url = "http://cs.pitt.edu/~wiebe/courses/CS0007/Lectures/hopedale.dat"
3 web_page = urllib.urlopen(url)
4 for line in web_page:
5     line = line.strip()
6     print(line)
7 web_page.close()
```

One Record per Line

Call the same function with an open web page instead of a local file:

```
1  import urllib
2  def process_file(filename):
3      '''Open, read, and print a file.'''
4      input_file = open(filename, "r")
5      for line in input_file:
6          line = line.strip()
7          print(line)
8  if __name__ == "__main__":
9      webpage = urllib.urlopen(sys.argv[1])
10     process_file(sys.argv[1])
11     webpage.close()
```

One Record per Line

Skipping the header. `tsdl.py`

```
1  def skip_header(r):
2      '''Skip the header in reader r, and return the first
3         real piece of data.'''
4
5      # Read the description line and then the comment lines.
6      line = r.readline()
7      line = r.readline()
8      while line.startswith('#'):
9          line = r.readline()
10     # Now line contains the first real piece of data.
11     return line
```

One Record per Line

Data with Missing Values: Fox Pelts

```
1  Coloured fox fur production, Hebron, Labrador, 1834-1839
2  #Source: C. Elton (1942) "Voles, Mice and Lemmings", Oxford Univ. Press
3  #Table 17, p.265--266
4  #remark: missing value for 1836
5  55
6  262
7  -
8  102
9  178
10 227
```

?,

One Record per Line

Find the smallest value

```
1  def smallest_value_skip(r):
2      line = skip_header(r).strip()
3      smallest = int(line)
4      for line in r:
5          line = line.strip()
6
7          if line != '-' : value = int(line)
8
9              if value < smallest:
10                 smallest = value
11     return smallest
```

One Record per Line

, , , \t, ...

Individual whitespace-delimited data

```
1  Annual Number of Lynx Trapped, MacKenzie River, 1821–1934
2  #Original Source: Elton, C. and Nicholson, M. (1942)
3  #"The ten year cycle in numbers of Canadian lynx",
4  #J. Animal Ecology, Vol. 11, 215––244.
5  #This is the famous data set which has been listed before in
6  #various publications:
7  #Cambell, M.J. and Walker, A.M. (1977) "A survey of statistical work on
8  #the MacKenzie River series of annual Canadian lynx trappings for the years
9  #1821–1934 with a new analysis", J.Roy.Statistical Soc. A 140, 432––436.
10 269. 321. 585. 871. 1475. 2821. 3928. 5943. 4950. 2577. 523.  98.
11 184. 279. 409. 2285. 2685. 3409. 1824. 409. 151.    45.  68. 213.
12 ...
13 485. 662. 1000. 1590. 2657. 3396.
```

One Record per Line

Individual whitespace-delimited data

- break each line into pieces
- and strip off the periods

```
For each other line of data:
    For each piece of data in the current line:
        Process that piece.
```

One Record per Line

Individual whitespace-delimited data

- Find the maximum value

```
Find the first line of real data after the header.
Find the largest value in that line.
For each other line of data:
    Find the largest value in that line.
    If that value is larger than the previous largest,
        remember it.
```

One Record per Line

Individual whitespace-delimited data

- Find the maximum value

```
1 def find_largest(line):
2     largest = -1
3     for value in line.split():
4         v = int(value[:-1])
5         if v > largest: largest = v
6     return largest
```

Records with Multiple Fields

United States housing data for 1983 and 1984

- the monthly housing starts(thousands of units)
- the total construction contracts (millions of dollars)
- the average interest rate for a new home mortgage (percent)

```
1  91.3    11.358    13
2  96.3    11.355    12.62
3  134.6   16.100    12.97
4  ...
5  98.9    14.220    12.05
```

Records with Multiple Fields

Store the data by column using two lists, monthly housing and construction contracts

```
1  import sys
2
3  def housing(r):
4      '''Return the difference between the housing starts and
5      construction contracts in 1983 and in 1984 from reader r.'''
6
7      # The monthly housing starts, in thousands of units.
8      starts = []
9
10     # The construction contracts, in millions of dollars.
11     contracts = []
12
13     # Read the file, populating the lists.
14     for line in r:
15         start, contract, rate = line.split()
16         starts.append(float(start))
17         contracts.append(float(contract))
18
19     return (sum(starts[12:24]) - sum(starts[0:12]),
20             sum(contracts[12:24]) - sum(contracts[0:12]))
21
22 if __name__ == "__main__":
23     input_file = open(sys.argv[1], "r")
24     print(housing(input_file))
25     input_file.close()
```

Records with Multiple Fields

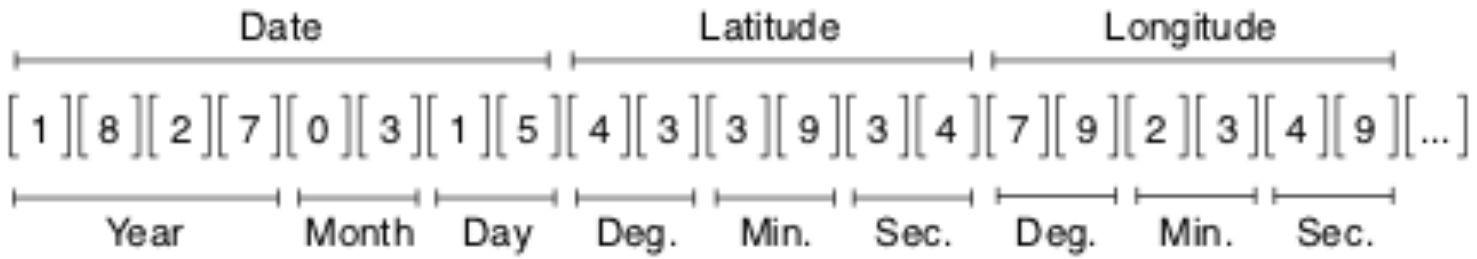
Design a function to separate the parsing and processing of the data

```
1  def read_housing_data(r):
2      '''Read housing data from reader r, returning lists of starts,
3      contracts, and rates.'''
4
5      starts = []
6      contracts = []
7      rates = []
8
9      for line in r:
10         start, contract, rate = line.split()
11         starts.append(float(start))
12         contracts.append(float(contract))
13         rates.append(rate)
14
15     return (starts, contracts, rates)
```

Positional Data

Some file formats don't use delimiters to separate fields.

- time(8 chars): year(4 chars), month(2 chars), day(2 chars)
- latitude(6 chars): degrees(2 chars), minutes(2 chars), seconds(2 chars)
- longitude(6 chars): degrees(2 chars), minutes(2 chars), seconds(2 chars)
- temperature(8 chars)
- humidity(8 chars)
- pressure(8 chars)



Positional Data

A function to parse fields using `slicing`

```
def read_weather_data(r):
    result = []
    for line in r:
        year = int(line[0:4]);
        month = int(line[4:6])
        day = int(line[6:8]);
        lat_deg = int(line[8:10])
        lat_min = int(line[10:12]);
        lat_sec = int(line[12:14])
        long_deg = int(line[14:16])
        long_min = int(line[16:18])
        long_sec = int(line[18:20])
        temp = float(line[20:26])
        hum = float(line[26:32])
        press = float(line[32:38])
        result.append(((year, month, day),
                       (lat_deg, lat_min, lat_sec),
                       (long_deg, long_min, long_sec),
                       (temp, hum, press)))
    return result
```

Positional Data

A function to parse fields using `slicing`

```
1  def read_weather_data(r):
2      fields = ((4, int), (2, int), (2, int),        # date
3                (2, int), (2, int), (2, int),        # latitude
4                (2, int), (2, int), (2, int),        # longitude
5                (6, float), (6, float), (6, float)) # data
6      result = []
7      # For each record
8      for line in r:
9          start = 0
10         record = []
11         # for each field in the record
12         for (width, target_type) in fields:
13             # convert the text
14             text = line[start:start+width]
15             field = target_type(text)
16             # add it to the record
17             record.append(field)
18             # move on
19             start += width
20         # add the completed record to the result
21         result.append(record)
22     return result
```

Positional Data

A function to parse fields using `slicing`

- The basic idea is that each field is a fixed width.
- We store a reference to the appropriate conversion function right beside each field's width.

```
1 fields = ((4, int), (2, int), (2, int),       # date
2           (2, int), (2, int), (2, int),       # latitude
3           (2, int), (2, int), (2, int),       # longitude
4           (6, float), (6, float), (6, float)) # data
```

Multiline Records

Every data record will not fit onto a single line.
Protein Data Bank(PDB) format that describes the arrangements of atoms in ammonia, `ammonia.pdb`

```
COMPND      AMMONIA
ATOM      1   N   0.257   -0.363    0.000
ATOM      2   H   0.257    0.727    0.000
ATOM      3   H   0.771   -0.727    0.890
ATOM      4   H   0.771   -0.727   -0.890
END
```

- the name of the molecule
- ID
- Type
- XYZ coordinates

Multiline Records

We have more molecules.

```
1  COMPND      AMMONIA
2  ATOM      1   N   0.257   −0.363    0.000
3  ATOM      2   H   0.257    0.727    0.000
4  ATOM      3   H   0.771   −0.727    0.890
5  ATOM      4   H   0.771   −0.727   −0.890
6  END
7  COMPND      METHANOL
8  ATOM      1   C   −0.748   −0.015    0.024
9  ATOM      2   O   0.558    0.420   −0.278
10 ATOM      3   H   −1.293   −0.202   −0.901
11 ATOM      4   H   −1.263    0.754    0.600
12 ATOM      5   H   −0.699   −0.934    0.609
13 ATOM      6   H   0.716    1.404    0.137
14 END
```

Multiline Records

How to read multiline records?

```
while there are more molecules in the file:
    read a molecule from the file
    append it to the list of molecules read so far
```

Multiline Records

```
1  def read_all_molecules(r):
2      '''Read zero or more molecules from reader r,
3      returning a list of the molecules read.'''
4
5      result = []
6      reading = True
7      while reading:
8          molecule = read_molecule(r)
9          if molecule:
10             result.append(molecule)
11         else:
12             reading = False
13     return result
```

Multiline Records

```
1  def read_molecule(r):
2      '''Read a single molecule from reader r and return it,
3      or return None to signal end of file.'''
4
5      # If there isn't another line, we're at the end of the file.
6      line = r.readline()
7      if not line:
8          return None
9
10     # Name of the molecule: "COMPND   name"
11     key, name = line.split()
12
13     # Other lines are either "END" or "ATOM num type x y z"
14     molecule = [name]
15     reading = True
16
17     while reading:
18         line = r.readline()
19         if line.startswith('END'):
20             reading = False
21         else:
22             key, num, type, x, y, z = line.split()
23             molecule.append((type, x, y, z))
24
25     return molecule
```

Looking Ahead

Suppose that molecules didn't have `END` markers but instead just a `COMPND` line followed by one or more `ATOM` lines. `read_molecule(r)`:

- extract the molecule's name from the `COMPND` line
- read `ATOM` lines until it got another `COMPND` line

After reading the `COMPND` line line, the line isn't available for the next call.

Looking Ahead

We must always ”look ahead” one line.

```
1  def read_all_molecules(r):
2      '''Read zero or more molecules from reader r,
3      returning a list of the molecules read.'''
4
5      result = []
6      line = r.readline()
7      while line:
8          molecule, line = read_molecule(r, line)
9          result.append(molecule)
10     return result
```

Looking Ahead

We must always ”look ahead” one line.

```
1  def read_molecule(r, line):
2      '''Read a molecule from reader r.  The variable 'line'
3      is the first line of the molecule to be read; the result is
4      the molecule, and the first line after it (or the empty string
5      if the end of file has been reached).'''
6
7      fields = line.split()
8      molecule = [fields[1]]
9
10     line = r.readline()
11     while line and not line.startswith('COMPND'):
12         fields = line.split()
13         key, num, type, x, y, z = fields
14         molecule.append((type, x, y, z))
15         line = r.readline()
16
17     return molecule, line
```

Writing to Files

How to create or modify files?

- r read mode (default)
- w write mode
- a append mode

```
1 output_file = open("test.txt", "a")
2 output_file.write("Computer Science")
3 output_file.close()
```

Writing to Files

What happen if we run `sum` function?

```
1  -110 300
2  18 78
3  90 -7
4  ...
```

```
1  def sum(input_file, output_filename):
2      output_file = open(output_filename, 'w')
3
4      for line in input_file:
5          operands = line.split()
6          print('operands', operands)
7          sum = float(operands[0]) + float(operands[1])
8          new_line = line.rstrip() + ' ' + str(sum) + '\n'
9          output_file.write(new_line)
10
11     output_file.close()
```

CSV Files I

- A comma-separated values (CSV) file is a delimited text file that uses a comma to separate values.
- Each line of the file is a data record.
- CSV files are so common that the Python standard library provides tools for working with them.

```
1  "Month", "Average", "2005", "2006", "2007", "2008",
2  "2009", "2010", "2011", "2012", "2013", "2014", "2015"
3  "May",  0.1,  0,  0, 1, 1, 0, 0, 0, 2, 0,  0,  0
4  "Jun",  0.5,  2,  1, 1, 0, 0, 1, 1, 2, 2,  0,  1
5  "Jul",  0.7,  5,  1, 1, 2, 0, 1, 3, 0, 2,  2,  1
6  "Aug",  2.3,  6,  3, 2, 4, 4, 4, 7, 8, 2,  2,  3
7  "Sep",  3.5,  6,  4, 7, 4, 2, 8, 5, 2, 5,  2,  5
8  "Oct",  2.0,  8,  0, 1, 3, 2, 5, 1, 5, 2,  3,  0
9  "Nov",  0.5,  3,  0, 0, 1, 1, 0, 1, 0, 1,  0,  1
10 "Dec",  0.0,  1,  0, 1, 0, 0, 0, 0, 0, 0,  0,  1
```

CSV Files II

```
1  from csv import reader, writer
2
3  # Open the two csv files.
4  infile = open("hurricanes.csv",'r')
5  csvReader = reader(infile)
6
7  outfile = open("filtered.csv", "w")
8  csvWriter = writer(outfile)
9
10 # Add the list of column headers to the csv file.
11 headers = ["Month","Average"]
12 csvWriter.writerow(headers)
13
14 # Skip the row of column headers in the reader.
15 next(csvReader)
16
17 # Filter the rows of data.
18 for row in csvReader:
19     month = row[0]
20     avg = float(row[1])
21
22     newRow = [month, avg]
23     csvWriter.writerow(newRow)
24
25 infile.close()
26 outfile.close()
```

Working with Files and Directories

Python supports efficent ways for manipulating files and directories.

- Running operating system commands and applications
- Listing files
- How to test whether a filename is a standard file, a directory, or a link and extract the age and size of a file
- How to remove files and directories, copy and rename files
- Manipuating a complete filepath into the directory part and the filename part.
- Creating directories and moving around in directory trees and processing files

Running Applications I

Run any operating system commands and user applications:

- using module `commands`
- using module `subprocess`

Listing 1: Using module `commands`

```
1  >>> import commands
2  >>> failure, output = commands.getstatusoutput("ls")
3  >>> if failure:
4  ...     print('Failed!\n')
5  ...     sys.exit(1)
6  ...
7  >>> print(output)
8  02-www.key
9  DiveIntoPython
10 ETC-Python
11 ...
```

Running Applications II

Listing 2: Using `Popen` and `PIPE` of module `subprocess`

```
1
2 >>> from subprocess import Popen, PIPE
3 >>> p = Popen("ls", shell=True, stdout=PIPE)
4 >>> output, errors = p.communicate()
5 >>> print(output)
6 02-www.key
7 DiveIntoPython
8 ETC-Python
9 ...
```

Listing Files I

How to get the data in a directory:

- Get a list of files in the current directory using module `glob`
- List all files in a director using module `os`

Listing 3: Using module `glob`

```
1 >>> import glob
2 >>> flist = glob.glob("*.key") + glob.glob("*.pdf")
3 >>> flist
4 ['02-www.key', 'Practical Programming.pdf']
```

Listing 4: Using module `os`

```
1 >>> import os
2 >>> files = os.listdir("/home/dshwang/dsLecture/Markup")
3 >>> curfiles = os.listdir(os.curdir)
4 >>> files
5 ['py-1.lyx~', '._lec09-03-inverting.tex', '._lec09-01-set.tex', 'py-2.lyx~', 'lec-09', 'lec
```

Testing File Types I

Test if a given filename is a regular file, a directory, or a link:

Listing 5: Using `isfile()`, `isdir()`, `islink()`

```
>>> import os.path
>>> myown = '02-www.key'
>>> if os.path.isfile(myown):
...     print('plain file')
... elif os.path.isdir(myown):
...     print('directory')
... elif os.path.islink(myown):
...     print('link')
... else:
...     print('nothing')
...
```

Measuring Time I

Time is measured in seconds since January 1, 1970.

Listing 6: Finding the age of a file and its size

```
>>> import os.path
>>> last_access = os.path.getatime('pyqtbook26.tar.gz')
>>> last_modification = os.path.getmtime('pyqtbook26.tar.gz')
>>> size = os.path.getsize('pyqtbook26.tar.gz')
>>> print(size, last_modification)
736810 1308128648.47
```

Get File Permission I

The `os.access` function tests read, write, and execute permissions of a file.

```
>>> import time, os
>>> myfile = 'pyqtbook26.tar.gz'
>>> if os.access(myfile, os.R_OK): # os.X_OK, os.W_OK
...     print(myfile, ' is read permission')
pyqtbook26.tar.gz  is read permission
>>>
```

Remove Files and Directories I

The `os.remove` function removes a single file.

```
1 >>> import os
2 >>> myfile = 'pyqtbook26.tar.gz'
3 >>> os.remove(myfile)
4 >>>
```

Removal of a collection of files is done with the `glob.glob` function.

```
1 >>> import os, glob
2 >>> for file in glob.glob('*.ps') + glob.glob('*.gif'):
3 ...     os.remove(file)
4 >>>
```

Remove Files and Directories I

Make a function `remove` for unified treatment of file and directory removal.

```
1 remove('my.dat') # remove a single file my.dat
2 remove('mytree') # remove a single directory tree mytree
3
4 # remove several files/trees with names in a list of strings:
5 remove(glob.glob('*.tmp') + glob.glob('*.temp'))
6 remove(['my.dat','mydir','yourdir'] + glob.glob('*.data'))
```

Removal of a collection of files is done with the `glob.glob` function.

```
1 >>> import os, glob
2 >>> for file in glob.glob('*.ps') + glob.glob('*.gif'):
3 ...     os.remove(file)
4 >>>
```

Remove Files and Directories I

Make a function `remove` for unified treatment of file and directory removal.

```
remove('my.dat') # remove a single file my.dat
remove('mytree') # remove a single directory tree mytree

# remove several files/trees with names in a list of strings:
remove(glob.glob('*.tmp') + glob.glob('*.temp'))
remove(['my.dat','mydir','yourdir'] + glob.glob('*.data'))
```

Listing 7: Function `remove`

```
def remove(files):
    """Remove one or more files and/or directories."""

    if isinstance(files, str): # is files a string?
        files = [files] # convert files from a string to a list
    if not isinstance(files, list): # is files not a list?
        print(files, " is not a list")
    for file in files:
        if os.path.isdir(file):
            shutil.rmtree(file)
        elif os.path.isfile(file):
            os.remove(file)
```

Copy and Rename Files I

Copying files is done with the `shutil` module.

```
1  import shutil
2
3  # copy a file into another
4  shutil.copy(myfile, tmpfile)
5
6  # copy last access time and last modification time
7  shutil.copy2(myfile, tmpfile)
8
9  # copy a directory tree
10 shutil.copytree(root_of_tree, destination_dir, True)
```

Cross-platform composition of pathnames

- `os.path.join` joins directory and file names with the right delimiter
- Variables `os.curdir` and `os.pardir` represent the current working directory and its parent directory.

```
1 shutil.copy(os.path.join(os.pardir,os.pardir,'f1.c'),
2 os.curdir)
```

The `os.rename` function is used to rename a file:

Copy and Rename Files II

```
1 os.rename(myfile, 'tmp.1') # rename myfile to 'tmp.1'
```

Splitting Pathnames I

Split a filepath into the basename and the directory name

```
>>> import os
>>> fname = '/home/dshwang/dsStudy/pyqtbook26.tar.gz'
>>> basename = os.path.basename(fname)
>>> dirname = os.path.dirname(fname)
>>> print(basename)
pyqtbook26.tar.gz
>>> print(dirname)
/home/dshwang/dsStudy
>>> # or
...
>>> dirname, basename = os.path.split(fname)
>>> print(dirname)
/home/dshwang/dsStudy
>>> print(basename)
pyqtbook26.tar.gz
```

The extension is extracted by the `os.path.splitext` function.

```
>>> root, extension = os.path.splitext(fname)
>>> print(root)
/home/dshwang/dsStudy/pyqtbook26.tar
>>> print(extension)
.gz
```

Splitting Pathnames II

Changing some arbitrary extension of a filename `f` to a new extension `ext` can be done by:

```
1 >>> import os
2 >>> fname = '/home/dshwang/dsStudy/pyqtbook26.tar.gz'
3 >>> new_fname = os.path.basename(os.path.splitext(fname)[0]+'.zip')
4 >>> new_fname
5 'pyqtbook26.tar.zip'
```

Creating and Moving Directories

Use the function `os.mkdir` for creating directories and `os.chdir` for browsing directories.

```
1 origdir = os.getcwd() # where we are
2 newdir = os.path.join(os.pathdir, 'mynewdir')
3 if not os.path.isdir(newdir):
4     os.mkdir(newdir) # or os.mkdir(newdir, '0755')
5 os.chdir(newdir)
6
7 os.chdir(os.environ['HOME']) # move to home directory
```

The function `os.makedirs` creates the whole path in one statement.

```
1 # $HOME/py/src/test1
2 os.makedirs(os.path.join(os.environ['HOME'],'py','src','test1'))
```

Summary

- Data stored in files is usually formatted in various ways.
- IO processing work should be broken into input, processing, and output stages.
- Programs that take filenames as command-line arguments are more flexible.
- Files can be read (content retrieved), written to (content replaced), and added to (new content appended).
- Data files come in many different formats. We had better provide helper functions.
- To make the functions usable by different types of readers, the reader is opened outside the function, passed as an argument to the function, and then closed outside the function.
- We reviewed various commands for working with files and directories.