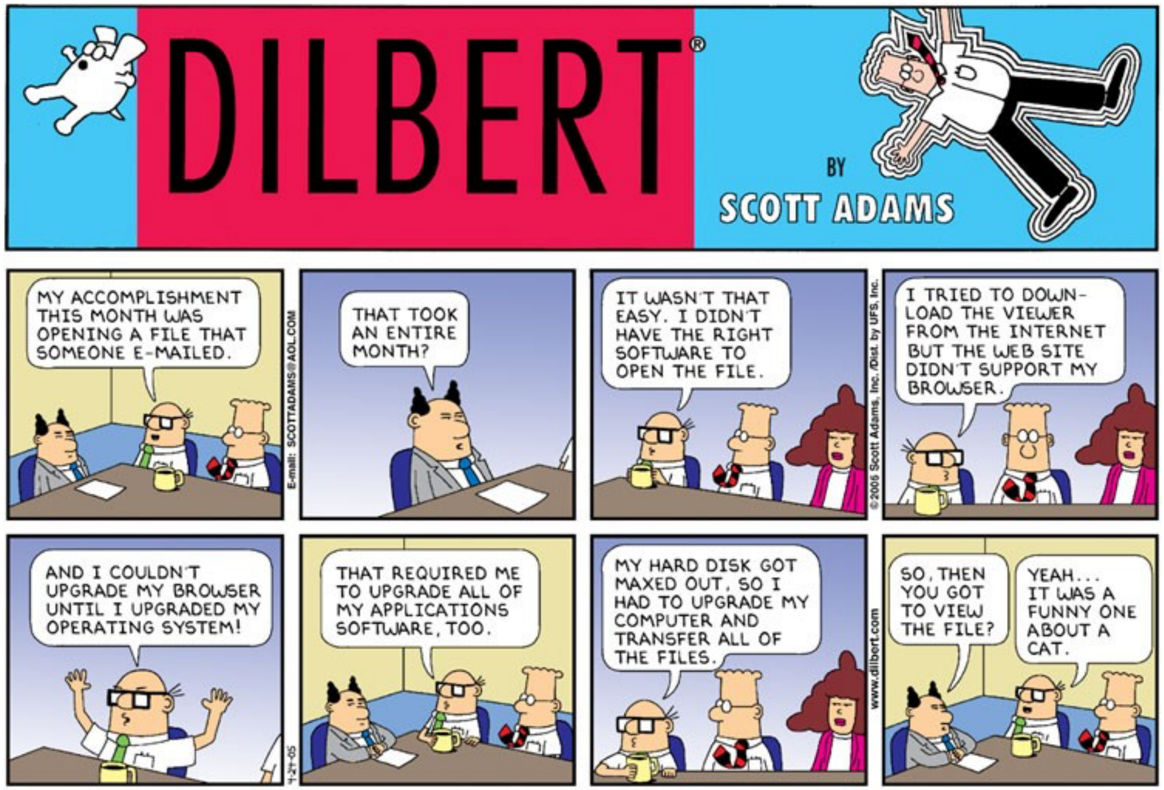
File Input/Output in Python

October 9, 2017


DILBERT®
BY SCOTT ADAMS
MY ACCOMPLISHMENT THIS MONTH WAS OPENING A FILE THAT SOMEONE E-MAILED.
E-mail: SCOTTADAMS@AOL.COM
THAT TOOK AN ENTIRE MONTH?
IT WASN'T THAT EASY. I DIDN'T HAVE THE RIGHT SOFTWARE TO OPEN THE FILE.
© 2006 Scott Adams, Inc. /Dist. by UFS, Inc.
I TRIED TO DOWNLOAD THE VIEWER FROM THE INTERNET BUT THE WEB SITE DIDN'T SUPPORT MY BROWSER.
AND I COULDN'T UPGRADE MY BROWSER UNTIL I UPGRADED MY OPERATING SYSTEM!
4-24-05
THAT REQUIRED ME TO UPGRADE ALL OF MY APPLICATIONS SOFTWARE, TOO.
MY HARD DISK GOT MAXED OUT, SO I HAD TO UPGRADE MY COMPUTER AND TRANSFER ALL OF THE FILES.
www.dilbert.com
SO, THEN YOU GOT TO VIEW THE FILE?
YEAH... IT WAS A FUNNY ONE ABOUT A CAT.

© Scott Adams, Inc./Dist. by UFS, Inc.

Moving beyond simple analysis – Use real data

- Most of you will have datasets that you want to do some analysis with (from simple statistics on few hundred sample points to millions of waveform files)

- This data is likely already contained in an existing file

- Can simply import it using range of python tools, able to use previously discussed python tools for analysis

Moving beyond simple analysis – Use real data

- Once analysis is done, you will want to capture result of analysis
  - Figures (use matplotlib)
  - Output data files

- Today we will cover a range of techniques for importing data files into python and exporting information into new files

- Lab today gives some examples/practice as well as have you create more “useful” deliverables based on input/output of files

File I/O Tools Covered:

- Built-in python
  - open(), read(), write(), close()
- NumPy
  - loadtxt(), genfromtxt(), savetxt()
- Pandas
  - read_csv(), read_excel(), to_csv()

Side Note: Commonly used file formats (for geoscience-type folks)

- Ascii or text files are those that are readable by humans
  - Create these in your text editor of choice
  - Sometimes called “flat file”
  - Has little/no formatting (no **bold**, *italics*, etc)
- Binary files – non-text file (not human readable), computer readable
  - Contain sequence of bytes grouped in eights
  - Compiled code = executable, example of binary file
  - Excel file format – another example of binary file
  - Opening in text editor show unintelligible characters
  - Lots of other file formats covering images, audio, software specific, etc
- Work with python tools that allow for use of both ascii and binary file formats

Delimiters

- Text files will have some way to indicate new columns of data, rows separated by newlines

- Range of these, with common examples:
  - Single white space
  - Tab
  - Comma
  - Colon

- Need to be aware of these as you read and write your own files

User Input and Simple Output on Screen

- In some cases, it’s useful to request some input from user (filename, range of parameters, etc)
- raw_input() and input() functions will do this, have differences in behavior in Python 2.7 (version we are using for class)
    - raw_input() assumes strings (will convert numbers to strings)
    - input() will evaluate whatever is in argument (can be numbers, functions, etc)

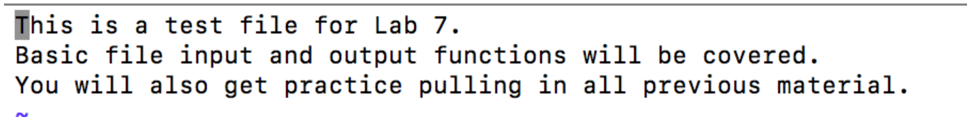
>>> txt = raw_input("Enter text here: ") #will print out text contained in “ “ onto screen, will wait for user response, put response into txt

- print() – simple printing. If no “file=“ parameter set, will simply print to screen

Built-in Python Functions: open(), read(), write(), close()

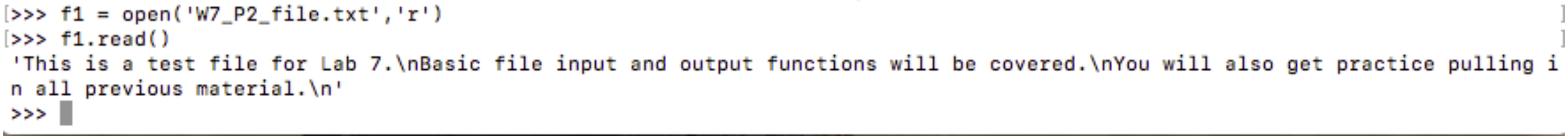
- Simple tools – no need to call special packages

- Example file: (has unseen line breaks)



```
This is a test file for Lab 7.
Basic file input and output functions will be covered.
You will also get practice pulling in all previous material.
~
```

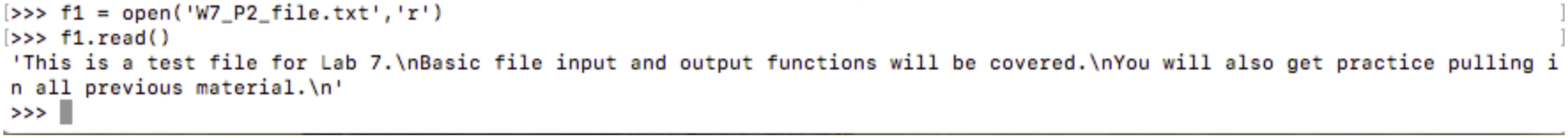
- Results can be ugly (print out the line breaks as \n)



```
[>>> f1 = open('W7_P2_file.txt','r')                                                                          ]
[>>> f1.read()                                                                                                ]
'This is a test file for Lab 7.\nBasic file input and output functions will be covered.\nYou will also get practice pulling i
n all previous material.\n'
>>> 
```

Built-in Python Functions: open(), read(), write(), close()

- Important steps – need to open file first before reading or writing



```
[>>> f1 = open('W7_P2_file.txt','r')                                                                      ]
[>>> f1.read()                                                                                            ]
 'This is a test file for Lab 7.\nBasic file input and output functions will be covered.\nYou will also get practice pulling i
 n all previous material.\n'
 >>> 
```

  - ‘r’: reading only
  - ‘r+’: reading and writing
  - ‘w’: writing only
  - ‘a’: append to end of existing file
  - ‘b’: use for binary files

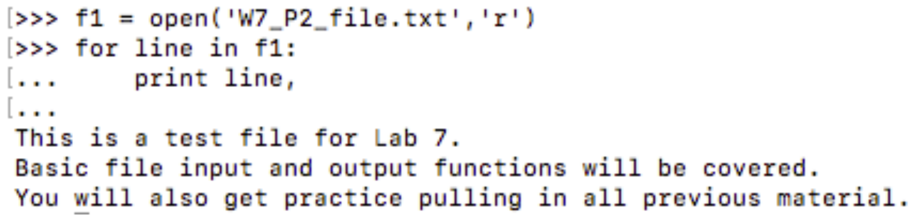
```
>>> f2 = open('file_write.txt','w')                 #opens a file for writing only
>>> f2.write('Practice at writing to a new file')   #puts text into the f2 file (here file_write.txt)
```

- Need to close file when done writing (makes sure that what you write in file actually gets written, done at close)

```
>>> f2.close()
```

Once you have a file open for reading….

- read()
    - Will read entire file’s contents at once
    - Ok for some purposes, but if you want to do something to each line, need something else
- readline()
    - Reads a single line in the file
    - Can do it multiple times for multiple lines, but
- Better option is looping over the file using for loop



```
[>>> f1 = open('W7_P2_file.txt','r')
[>>> for line in f1:
[...     print line,
[...
 This is a test file for Lab 7.
 Basic file input and output functions will be covered.
 You will also get practice pulling in all previous material.
```

NumPy: loadtxt(), genfromtxt(), savetxt()

- Remember NumPy library useful for dealing with arrays
- Can use NumPy tools to read and write files, easily put data into NumPy arrays
- Need to remember to import the NumPy library before using
- >> import numpy as np

Loading files with NumPy

- loadtxt(): simplest, can define filename (here also skipping row 1), puts all data into a NumPy array

```
>>> np.loadtxt('data_table.txt', skiprows=1)
array([[ 0.2536, 0.1008, 0.3857],
[ 0.4839, 0.4536, 0.3561],
[ 0.1292, 0.6875, 0.5929],
[ 0.1781, 0.3049, 0.8928],
[ 0.6253, 0.3486, 0.8791]])
```

- Other useful parameters
  - usecols= to specify which columns to read
  - unpack=True to split into multiple arrays
  - delimiter= ‘,’ to define the delimiter (white space is default)

- Some issues
  - Default data type is float, need to specify if not for each column
  - Files with missing data cause errors

Loading files with NumPy

- genfromtxt()
  - More flexible way to import data into NumPy array
  - Very useful parameter: dtype -- if use “=None”, will be assigned by what’s in each column
  - Can define how to handle missing data (define “missing_values” and “filling_values”)
  - Examples below pulls all data from space-delimited file file called ‘station.txt’,
    - skipping the header line

```
>>>  example_array = np.genfromtxt('station.txt', dtype=None, delimiter=" ", skip_header=1)
```

    - use column names in first line to define names of columns in array (access using these names)

```
>>>example_array = np.genfromtxt('station.txt', dtype=None, delimiter=" ", names=True)
```

NumPy: loadtxt() vs genfromtxt()

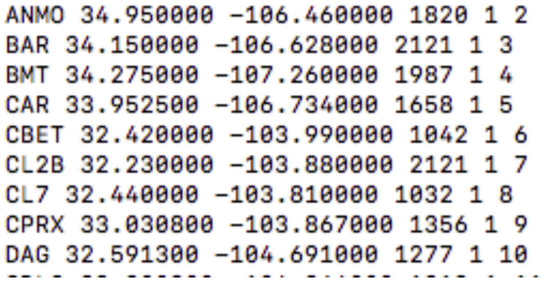
- Which one to use?
  - genfromtxt() is better option unless you have very simple files
  - Supports many of the same parameters as loadtxt(), but handles missing data and defines data type automatically, which is useful if you have some columns with strings

NumPy Saving Files

- savetxt(): saves an array to a text file
  - Need to define an output filename and the array to save
  - Can also define the format of the array objects, delimiters, header, footer, comments

```
>>> np.savetxt('file.out', example_array, fmt='%s %f %f %i %i %i')
```

Saves the example_array to a file called file.out:



```
ANMO 34.950000 -106.460000 1820 1 2
BAR 34.150000 -106.628000 2121 1 3
BMT 34.275000 -107.260000 1987 1 4
CAR 33.952500 -106.734000 1658 1 5
CBET 32.420000 -103.990000 1042 1 6
CL2B 32.230000 -103.880000 2121 1 7
CL7 32.440000 -103.810000 1032 1 8
CPRX 33.030800 -103.867000 1356 1 9
DAG 32.591300 -104.691000 1277 1 10
```

Pandas: read_csv(), read_excel(), to_csv()

- Covered pandas Series and DataFrames last week – very useful data structures that can be manipulated with various functions in numpy and pandas

- Can also read data from files directly into these structures, using a variety of text and binary formatted files (including MS Excel)

- Can write these structures back out into text files (also excel files, but why?)

- Functions contained within the pandas library, so need to import that before using

```
>>> import pandas as pd
```

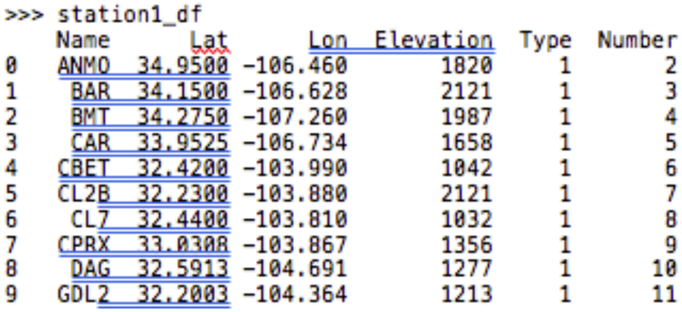
Pandas: read_csv()

- Use follows previous examples

  >>> station1_df = pd.read_csv('station.txt', sep=" ", header=0)

  Using the header=0 will pull the names of columns in first line of file to be used as names in DataFrame

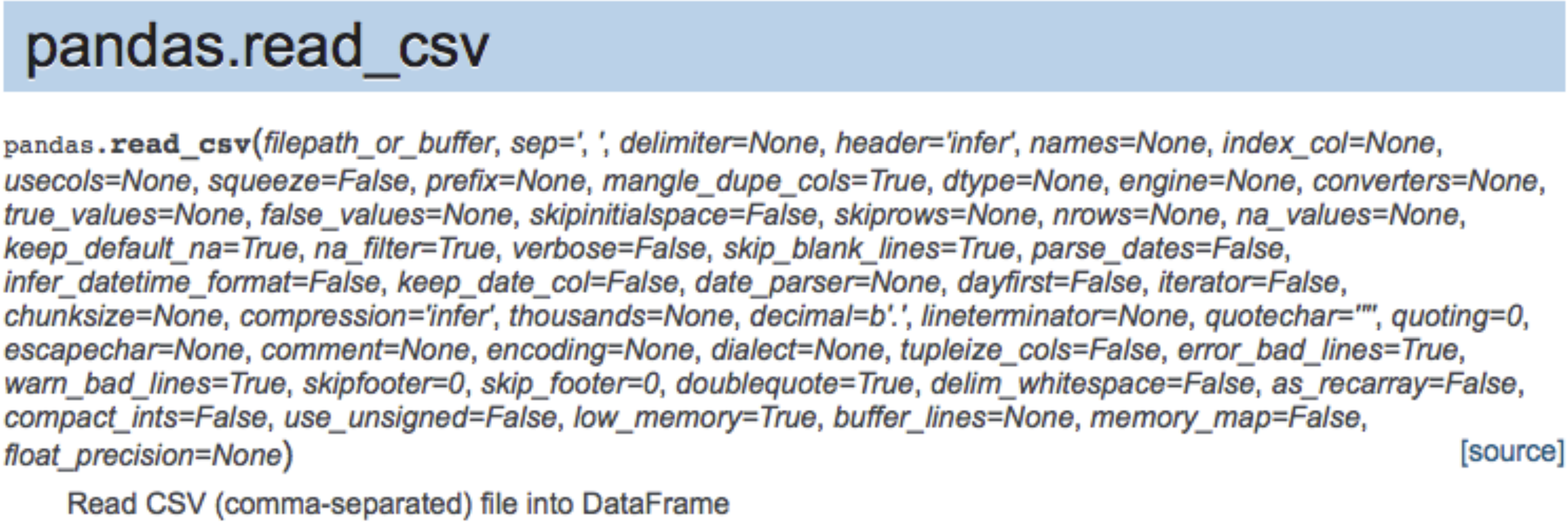
- Gives a data structure (station1_df) that contains data from the ’station.txt’ file



```
>>> station1_df
    Name      Lat      Lon  Elevation  Type  Number
0   ANMO  34.9500 -106.460       1820     1       2
1    BAR  34.1500 -106.628       2121     1       3
2    BMT  34.2750 -107.260       1987     1       4
3    CAR  33.9525 -106.734       1658     1       5
4   CBET  32.4200 -103.990       1042     1       6
5   CL2B  32.2300 -103.880       2121     1       7
6    CL7  32.4400 -103.810       1032     1       8
7   CPRX  33.0308 -103.867       1356     1       9
8    DAG  32.5913 -104.691       1277     1      10
9   GDL2  32.2003 -104.364       1213     1      11
```

Pandas: read_csv()

- A lot of options available to customize the reading (filename is required):



pandas.read_csv

```
pandas.read_csv(filepath_or_buffer, sep=', ', delimiter=None, header='infer', names=None, index_col=None,
usecols=None, squeeze=False, prefix=None, mangle_dupe_cols=True, dtype=None, engine=None, converters=None,
true_values=None, false_values=None, skipinitialspace=False, skiprows=None, nrows=None, na_values=None,
keep_default_na=True, na_filter=True, verbose=False, skip_blank_lines=True, parse_dates=False,
infer_datetime_format=False, keep_date_col=False, date_parser=None, dayfirst=False, iterator=False,
chunksize=None, compression='infer', thousands=None, decimal=b'.', lineterminator=None, quotechar='"', quoting=0,
escapechar=None, comment=None, encoding=None, dialect=None, tupleize_cols=False, error_bad_lines=True,
warn_bad_lines=True, skipfooter=0, skip_footer=0, doublequote=True, delim_whitespace=False, as_recarray=False,
compact_ints=False, use_unsigned=False, low_memory=True, buffer_lines=None, memory_map=False,
float_precision=None)
```

[source]

Read CSV (comma-separated) file into DataFrame

Pandas: read_excel()

- Another great tool if you have data in MS Excel files and don’t want to open in Excel to save as a text file.

- Very similar to read_csv() except need to define the sheetname to read

```
>>> station2_df = pd.read_excel('station.xlsx', sheetname='Sheet1')
```

Saving DataFrame to File: to_csv

- Similar structure to previous examples – will save a DataFrame to textfile

```
>>> station1_df.to_csv('station_out.txt', sep=" ")
```

Summary:

- There is a wide range of file input/output tools available in python to handle user input, text and binary files (like MS Excel)

- Which one you select depends a lot on
  - Input file format, data type, and complexity
  - What analysis tools you need for the data

Think about a data file that you have for your research:

- Write down the basic structure of the file (how many columns, rows, what do they consist of (strings, floats, int for example)

- What kind of analysis do you need to (or want to, or could) do with data in this file?

- What input/output tool would work best for that purpose?

- Bring this (+ file) to lab today – have an opportunity to try it!

Next up:

- Lab today: practice with file input and output, as well as bring together the variety of tools covered so far

- Next week: Leave python to cover some basic UNIX tools