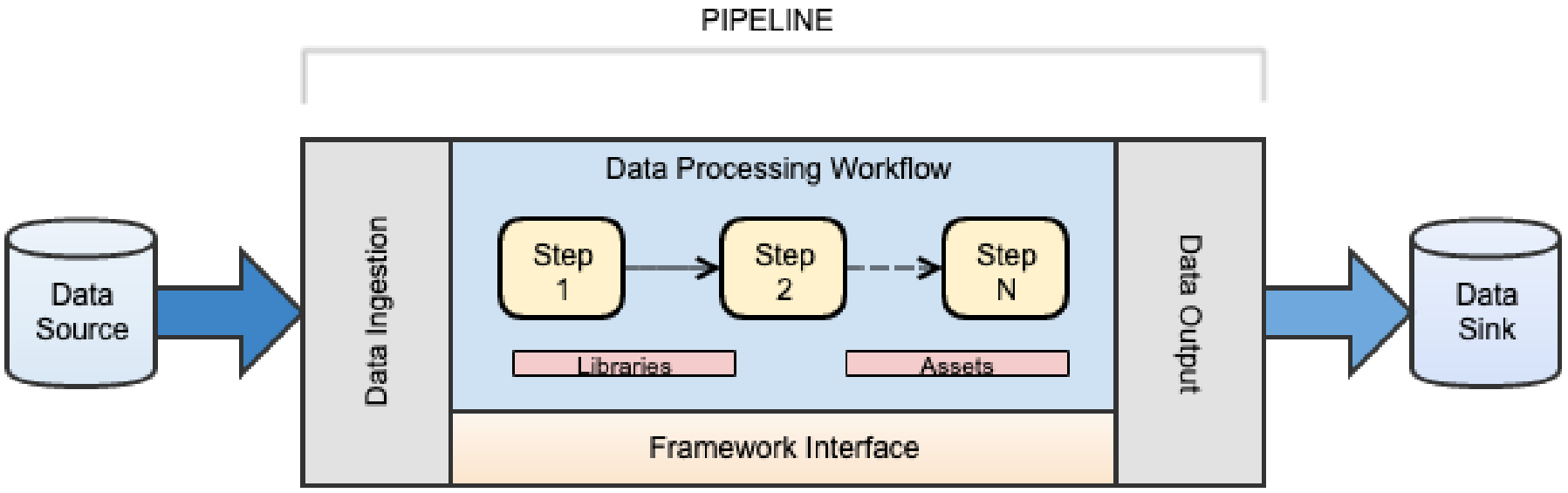
GEOS 436 / 636
Programming and Automation for Geoscientists

– Week 06: Data I/O –

Ronni Grapenthin
rgrapenthin@alaska.edu
Elvey 413B
x7682

How can you **process your data**
and
preserve the results of your work?

Data Processing Pipeline



here.com

So far, we've dealt with parts of the processing workflow.
Today, we add Data Input (Ingestion) and Output (I/O).

Data Storage

- Storage-needs range from a file with a handful of rows and columns to data distributed over millions of files each with massive amounts of data
- **Consistent** formatting of the data allows processing of MANY files with one program
- Consistent formatting is difficult for people to do (think hard about format initially!)
- Data are stored in a vast range of different data formats
- Main categories are: **text-based** (modern Excel `*.xlsx`, other tabulated data `*.csv(x), *.txt, ...`), and **binary** (old Excel `*.xls`, HDF5 `*.h5`)
- Main difference: text files are human-readable, binary files are not

Generic Data Ingestion

- To do anything worthwhile, you **MUST** know your data's format.
- Binary?
  - Is there a reader for that specific format (e.g., `*.xls`)?
  - What is the data structure that is being created by that reader?
  - How do I access data point `x`?
  - READ THE DOCUMENTATION!
- Text-based?
  - Is there a reader for that specific format (e.g., `*.xlsx, *.csv`)? See above.
  - Often (NOT always) file contains "header" information, one or more lines that explain how the data are formatted, what fields mean etc.
  - Are all "columns" numeric or are there also strings, other types?
  - Write your own reader for specific formats.
  - Modern tools (Numpy, Pandas) make that quite easy (one function call)

Generic Data Output 1/3

- Data output ranges widely:
  1. writing something to the screen
  2. storing numerical results in a file on disk
  3. make a plot
- Plotting is next week, we focus on (1) and (2)

Generic Data Output 2/3

Output to the screen can be highly effective

- You can read it (small volumes of output)
- You may be able to “pipe” the output into another program (more about that in unix tools)
- You can redirect it dynamically into a file.

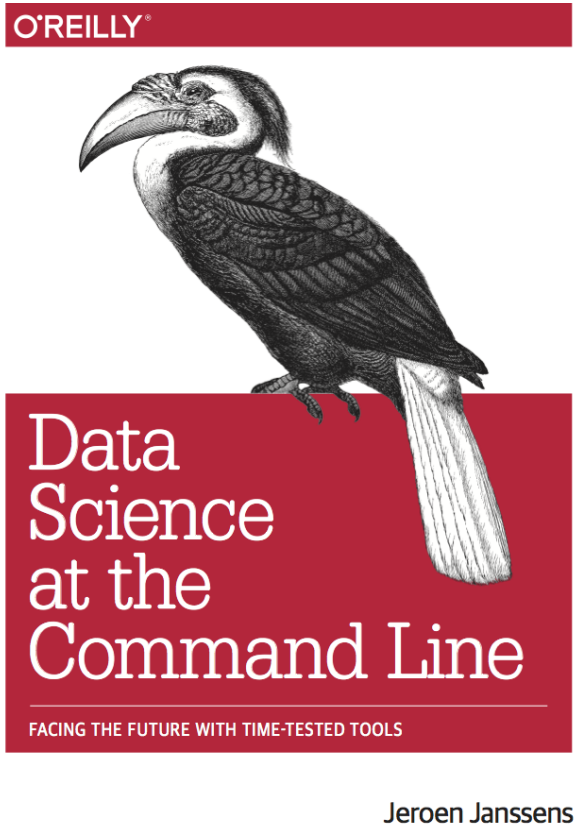
Generic Data Output 3/3

Output to files

- Should happen if the computations take a long time (modeling)
- Same things for data formats apply as for reading
- Do your best to keep documentation with data (e.g., header lines, self-documenting formats)
- It's worth to think about tradeoffs between:
  - long-term readability (text files will always be readable)
  - proprietary data formats (Will you be able to read an Excel file in 30 years?)
  - storage efficiency (Is it important that the file is 2MB vs 5MB, 500MB vs 5TB?)

I/O on the Commandline

- Listed for completeness. Will be covered at length during the Unix Tools sessions later in the semester.
- For now: **There are excellent, time-tested tools available for efficient processing of large quantities of (text-based) data.**



O'Reilly

Read the book for free at `https://www.datascienceatthecommandline.com/1e`

I/O in Base Python

Output to screen – `print()`

- We've used this quite a bit, Lab will go over some functionality.
- This is an incredibly important tool! Allows your program to give feedback to you.
- Put some time into exploring the capabilities.

```
#Basic print
print("(1) Hello World!")

#Doesn't just print strings
print("(2)", [23,2,"a"])

#Add some special formatting:
#Newline:       \n
#Tabulator:     \t
print("(3) A new line\n\tand a tab\n\tand another")

#number formatting (several possibilities)
print("(4) Float: %.2f and Int: %d" % (3.1415, 44))
```

```
(1) Hello World!
(2) [23, 2, 'a']
(3) A new line
        and a tab
        and another
(4) Float: 3.142 and Int: 44
```

I/O in Base Python

Input:

- Base Python comes with tools to read text, binary files: `open()`
- Can read full files or lines `read(), readlines()`
- Basic functionality, requires some work to get various files properly read

I/O in Base Python

```
fname = "io_print.txt"

#1) open this file in read mode
print("Example␣1")
print("--------------------Start")
my_file = open(fname, "r")
#print the entire thing
print(my_file.read())
#close the file
my_file.close()
print("--------------------End")

#2) print a two lines of the file
print("Example␣2")
print("--------------------Start")
my_file = open(fname, "r")
#read and print a line, twice
print(my_file.readline())
print(my_file.readline())
#close the file
my_file.close()
print("--------------------End")

#3) print each line in a loop
print("Example␣3")
print("--------------------Start")
with open(fname, 'r') as my_file:
    for l in my_file:
        print(l, end='')
print("--------------------End")
```

```
Example 1
--------------------Start
(1) Hello World!
(2) [23, 2, 'a']
(3) A new line
        and a tab
        and another
(4) Float: 3.142 and Int: 44

--------------------End
Example 2
--------------------Start
(1) Hello World!

(2) [23, 2, 'a']

--------------------End
Example 3
--------------------Start
(1) Hello World!
(2) [23, 2, 'a']
(3) A new line
        and a tab
        and another
(4) Float: 3.142 and Int: 44
--------------------End
```

I/O in Base Python

Output to file:

- Same as input (use `open()` in write `w`, or append `a` mode
- Functionality similarly basic, requires some work to get it right

I/O in Base Python

```
fname = 'io_write.txt'

#open file in write mode, add text
#needs newline if we want them
f = open(fname, "w")
f.write("First Text\n")
f.close()

#open file in write mode again, overwrite
f = open(fname, "w")
f.write("Second Text\n")
f.close()

#open file in append mode and append text
f = open(fname, "a")
f.write("Appended Text\n")
f.close()
```

```
Second Text
Appended Text
```

File Handling with NumPy

Input and output

NumPy binary files (NPY, NPZ)

load(file[, mmap_mode, allow_pickle, ...])	Load arrays or pickled objects from `.npy`, `.npz` or pickled files.
save(file, arr[, allow_pickle, fix_imports])	Save an array to a binary file in NumPy `.npy` format.
savez(file, *args, **kwds)	Save several arrays into a single file in uncompressed `.npz` format.
savez_compressed(file, *args, **kwds)	Save several arrays into a single file in compressed `.npz` format.

The format of these binary file types is documented in numpy.lib.format

Text files

loadtxt(fname[, dtype, comments, delimiter, ...])	Load data from a text file.
savetxt(fname, X[, fmt, delimiter, newline, ...])	Save an array to a text file.
genfromtxt(fname[, dtype, comments, ...])	Load data from a text file, with missing values handled as specified.
fromregex(file, regexp, dtype[, encoding])	Construct an array from a text file, using regular expression parsing.
fromstring(string[, dtype, count, sep])	A new 1-D array initialized from text data in a string.
ndarray.tofile(fid[, sep, format])	Write array to a file as text or binary (default).
ndarray.tolist()	Return the array as an `a.ndim`-levels deep nested list of Python scalars.

Raw binary files

fromfile(file[, dtype, count, sep, offset])	Construct an array from data in a text or binary file.
ndarray.tofile(fid[, sep, format])	Write array to a file as text or binary (default).

numpy.org

File Handling with NumPy

Input and output

NumPy binary files (NPY, NPZ)

load(file[, mmap_mode, allow_pickle, ...])	Load arrays or pickled objects from `.npy`, `.npz` or pickled files.
save(file, arr[, allow_pickle, fix_imports])	Save an array to a binary file in NumPy `.npy` format.
savez(file, *args, **kwds)	Save several arrays into a single file in uncompressed `.npz` format.
savez_compressed(file, *args, **kwds)	Save several arrays into a single file in compressed `.npz` format.

The format of these binary file types is documented in numpy.lib.format

Text files

loadtxt(fname[, dtype, comments, delimiter, ...])	Load data from a text file.
savetxt(fname, X[, fmt, delimiter, newline, ...])	Save an array to a text file.
genfromtxt(fname[, dtype, comments, ...])	Load data from a text file, with missing values handled as specified.
fromregex(file, regexp, dtype[, encoding])	Construct an array from a text file, using regular expression parsing.
fromstring(string[, dtype, count, sep])	A new 1-D array initialized from text data in a string.
ndarray.tofile(fid[, sep, format])	Write array to a file as text or binary (default).
ndarray.tolist()	Return the array as an `a.ndim`-levels deep nested list of Python scalars.

Raw binary files

fromfile(file[, dtype, count, sep, offset])	Construct an array from data in a text or binary file.
ndarray.tofile(fid[, sep, format])	Write array to a file as text or binary (default).

numpy.org

Read Textfile with NumPy - genfromtxt

numpy.genfromtxt

numpy.**genfromtxt**(*fname, dtype=<class 'float'>, comments='#', delimiter=None, skip_header=0, skip_footer=0, converters=None, missing_values=None, filling_values=None, usecols=None, names=None, excludelist=None, deletechars=" !#$%&'()*+, -./:;<=>?@[\]^{|}~", replace_space='_', autostrip=False, case_sensitive=True, defaultfmt='f%i', unpack=None, usemask=False, loose=True, invalid_raise=True, max_rows=None, encoding='bytes'*) [source]

Load data from a text file, with missing values handled as specified.

Each line past the first *skip_header* lines is split at the *delimiter* character, and characters following the *comments* character are discarded.

Parameters:

fname : *file, str, pathlib.Path, list of str, generator*

File, filename, list, or generator to read. If the filename extension is *gz* or bz2, the file is first decompressed. Note that generators must return byte strings. The strings in a list or produced by a generator are treated as lines.

dtype : *dtype, optional*

Data type of the resulting array. If None, the dtypes will be determined by the contents of each column, individually.

comments : *str, optional*

The character used to indicate the start of a comment. All the characters occurring on a line after a comment are discarded

delimiter : *str, int, or sequence, optional*

The string used to separate values. By default, any consecutive whitespaces act as delimiter. An integer or sequence of integers can also be provided as width(s) of each field.

skiprows : *int, optional*

skiprows was removed in numpy 1.10. Please use *skip_header* instead.

skip_header : *int, optional*

The number of lines to skip at the beginning of the file.

skip_footer : *int, optional*

The number of lines to skip at the end of the file.

numpy.org

Read Textfile with NumPy - genfromtxt

```
import numpy as np

#Uses the header line to name
#the output field via "names=True"
stations = np.genfromtxt('station.txt',
                         encoding='utf-8',
                         dtype=None,
                         delimiter='␣',
                         names=True)

#what did you get back?
print(stations)

#what are the column names?
print(stations.dtype.names)

#get all station names
print(stations['Name'])

#get all station latitudes
print(stations['Lat'])
```

```
[('ANMO', 34.95  , -106.46 , 1820, 1,  2)
 ('BAR', 34.15  , -106.628, 2121, 1,  3)
 ('BMT', 34.275 , -107.26 , 1987, 1,  4)
 ('CAR', 33.9525, -106.734, 1658, 1,  5)
 ('CBET', 32.42  , -103.99 , 1042, 1,  6)
 ('CL2B', 32.23  , -103.88 , 2121, 1,  7)
 ('CL7', 32.44  , -103.81 , 1032, 1,  8)
 ('CPRX', 33.0308, -103.867, 1356, 1,  9)
 ('DAG', 32.5913, -104.691, 1277, 1, 10)
 ('GDL2', 32.2003, -104.364, 1213, 1, 11)
 ('HTMS', 32.47  , -103.6  , 1192, 1, 12)
 ('LAZ', 34.402 , -107.139, 1878, 1, 13)
 ('LEM', 34.166 , -106.972, 1698, 1,  1)
 ('LPM', 34.3117, -106.632, 1737, 1, 14)
 ('MLM', 34.81  , -107.145, 2088, 1, 15)
 ('SBY', 33.9752, -107.181, 3230, 1, 16)
 ('SMC', 33.7787, -107.019, 1560, 1, 17)
 ('SRH', 32.4914, -104.515, 1276, 1, 18)
 ('SSS', 32.35  , -103.41 , 1072, 1, 19)
 ('Y22A', 33.937 , -106.965, 1674, 1, 20)
 ('Y22D', 34.0739, -106.921, 1436, 1, 21)
 ('WTX', 34.0722, -106.946, 1555, 1, 22)]
('Name', 'Lat', 'Lon', 'Elevation', 'Type', 'N
['ANMO' 'BAR' 'BMT' 'CAR' 'CBET' 'CL2B' 'CL7'
 'LAZ' 'LEM' 'LPM' 'MLM' 'SBY' 'SMC' 'SRH' 'SS
[34.95   34.15   34.275  33.9525 32.42
32.23   32.44   33.0308 32.5913
 32.2003 32.47   34.402  34.166
34.3117 34.81   33.9752 33.7787 32.4914
 32.35   33.937  34.0739 34.0722]
```

Save Textfile with NumPy - savetxt

numpy.savetxt

numpy.**savetxt**(*fname, X, fmt='%.18e', delimiter=' ', newline='n', header='', footer='', comments='# ', encoding=None*) [source]

Save an array to a text file.

Parameters:

fname : *filename or file handle*

If the filename ends in `.gz`, the file is automatically saved in compressed gzip format. loadtxt understands gzipped files transparently.

X : *1D or 2D array_like*

Data to be saved to a text file.

fmt : *str or sequence of strs, optional*

A single format (%10.5f), a sequence of formats, or a multi-format string, e.g. 'Iteration %d – %10.5f', in which case *delimiter* is ignored. For complex *X*, the legal options for *fmt* are:

- a single specifier, *fmt='%.4e'*, resulting in numbers formatted like *' (%s+%sj)' % (fmt, fmt)*
- a full string specifying every real and imaginary part, e.g. *' %.4e %+.4ej %.4e %+.4ej %.4e %+.4ej'* for 3 columns
- a list of specifiers, one per column - in this case, the real and imaginary part must have separate specifiers, e.g. *['%.3e + %.3ej', '(%.15e%+.15ej)']* for 2 columns

delimiter : *str, optional*

String or character separating columns.

newline : *str, optional*

String or character separating lines.
New in version 1.5.0.

header : *str, optional*

String that will be written at the beginning of the file.
New in version 1.7.0.

footer : *str, optional*

String that will be written at the end of the file.
New in version 1.7.0.

comments : *str, optional*

String that will be prepended to the `header` and `footer` strings, to mark them as comments. Default: '# ', as expected by e.g. `numpy.loadtxt`.
New in version 1.7.0.

numpy.org

Read Textfile with NumPy - savetxt

Eliminate all but three entries from station database, make a new file:

```
import numpy as np

#Uses the header line to name the output variables, using "names=True"
stations = np.genfromtxt('station.txt', encoding='utf-8',
                         dtype=None, delimiter='␣', names=True)

#delete all but the first three entries
stations = np.delete(stations, range(3,len(stations)), 0)

np.savetxt('stations_short.txt', stations, fmt='%s␣%f␣%f␣%i␣%i␣%i')
```

New file contents:

```
ANMO 34.950000 -106.460000 1820 1 2
BAR 34.150000 -106.628000 2121 1 3
BMT 34.275000 -107.260000 1987 1 4
```

File Handling with Pandas

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	Fixed-Width Text File	read_fwf	
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	ORC Format	read_orc	
binary	Msgpack	read_msgpack	to_msgpack
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	SPSS	read_spss	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google BigQuery	read_gbq	to_gbq

pandas.pydata.org

File Handling with Pandas

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	Fixed-Width Text File	read_fwf	
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	ORC Format	read_orc	
binary	Msgpack	read_msgpack	to_msgpack
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	SPSS	read_spss	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google BigQuery	read_gbq	to_gbq

pandas.pydata.org

Write CSV File with Pandas - read_csv

```
import pandas as pd

#get a dataframe that uses the first column as index (index_col=0)
#and has a header line, and field separator is a whitespace ' '
stations = pd.read_csv('station.txt', sep=' ', header=0, index_col=0)

#print three stations we are interested in, using the index
#(isn't that easy?)
print(stations.loc[['ANMO', 'BAR', 'CL7']])
```

```
        Lat      Lon  Elevation  Type  Number
Name
ANMO  34.95 -106.460       1820     1       2
BAR   34.15 -106.628       2121     1       3
CL7   32.44 -103.810       1032     1       8
```

Write CSV File with Pandas - to_csv

Select three entries from station database, make a new file:

```
import pandas as pd

#get a dataframe that uses the first column as index (index_col=0)
#and has a header line, and field separator is a whitespace ' '
stations = pd.read_csv('station.txt', sep=' ', header=0, index_col=0)

#select a subset of stations
new_stations = stations.loc[['ANMO', 'BAR', 'CL7']]

#write it to a new file
new_stations.to_csv('stations_short.csv')
```

New file contents:

```
Name,Lat,Lon,Elevation,Type,Number
ANMO,34.95,-106.46,1820,1,2
BAR,34.15,-106.628,2121,1,3
CL7,32.44,-103.81,1032,1,8
```