Python Workshop
File Input/Output

Mike Fienen

U.S. Geological Survey
Wisconsin Water Science Center, Middleton, Wisconsin USA

USGS National Groundwater Workshop, August 2012

science for a changing world

Outline

Overview

- Much of what is useful to do in Python is reading files, manipulating the data, and writing out results in another format
- Python and Numpy provide ways to read and write ASCII and binary files. We will focus on ASCII files

Background Information

http://waterservices.usgs.gov/
http://nwis.waterdata.usgs.gov/nwis/pmcodes/
http:
//docs.python.org/tutorial/inputoutput.html

Reading and Writing with Strings

- The simplest way to write information to a string is using `str`

```
>>>a = 5.4
>>>str(a)
'5.4'
```

- We typically want more control. Two main ways to do it. Old school (`%`) and new school (`format`)
- Formatted input and output are a key difference between Python 2.X and 3.X

Writing Strings the Old School Way (%)

- The general syntax is to make a string with conversion types for variables. For example:

```
>>>outstr = 'I have %21.1f kg of %s and %d bins of %s' %(3.99983,'eggs',53,'spam')
>>> outstr
'I have                   4.0 kg of eggs and 53 bins of spam'
>>>outstr = 'I have %-21.1f kg of %s and %0.3d bins of %s' %(3.99983,'eggs',53,'spam')
>>> outstr
'I have 4.0                   kg of eggs and 053 bins of spam'
```

- The general idea is to make a string including `'%'`, a conversion flag (optional), a width and resolution (optional), and a conversion type (required).

  For example: `%<flag><width>.<resolution><type>`
  `%-12.3f` Is a left-justified, floating point value with width of 12 and 3 decimal places.

- Following the format string must be a list of values as a tuple identified by `%`

  `'%4d is the %s\n' %(42,'answer')`

Writing Strings the Old School Way (`%`)

Details about formatted output available at:

`http://docs.python.org/library/stdtypes.html`

- Conversion flag characters

`'#'`	Invokes alternate behavior (see website for details)
`'0'`	Pads numeric values with zeros
`'-'`	Left-adjusts the output
`' '`	Leave a space before signed positive values so they line up with negative ones

USGS
science for a changing world

Writing Strings the Old School Way (%)

Details about formatted output available at:

http://docs.python.org/library/stdtypes.html

- Conversion flag characters

`'#'`	Invokes alternate behavior (see website for details)
`'0'`	Pads numeric values with zeros
`'-'`	Left-adjusts the output
`' '`	Leave a space before signed positive values so they line up with negative ones

- Most common conversion types.

`%d` or `%i`	Signed integer
`%f` or `%F`	Floating point
`%e` or `%E`	Floating point exponential (lower or upper case)
`%g` or `%G`	Combination of `%f` and `%e` depending on resolution
`%s` or `%r`	String. Width is used, but not resolution

USGS
science for a changing world

Writing Strings the New School Way (`format`)

Details about new school string formatting at:

`http://docs.python.org/library/string.html#formatstrings`

- The general syntax is similar, but conversion information is supplied differently. For example:

```
>>>outstr = 'I have %{0:21.1f} kg of {1:s} and {2:0=3d} bins of {3:s}'.format(3.99983,'
>>> outstr
'I have                    4.0 kg of eggs and 053 bins of spam'
```

- In this case, make a string including {...}, statements with conversion information.
  The general pattern is {`[index]`:`[format]`}
  - The `[index]` argument refers to the item index being mapped in
  - The `[format]` argument is similar to those in the old school way, but with some additional flexibility

Now that we can write strings, how about we write them to files?
We should also know how to read stuff back in from files

Interacting with Text Files

- The first thing is to open a file and make a file object

```
ifp = open('somefile.txt','r')
ofp = open('someotherfile.txt','w')
```

  - This object can be used to read or write from.
    I use `ifp` for “input file pointer” and `ofp` for “output file pointer”
    The arguments `'r'` and `'w'` indicate “read” and “write” respectively.

- To read the file can use `readline()` or `readlines()`
  - The difference is that `readlines()` reads the entire file into memory rather than `readline()` which reads one line at a time. Most of the time, `readlines()` is better
  - With `readlines()` once the data are read in, the result is a list with each element representing a line in the text file

`np.genfromtxt`: flexible way to read columns

Example file: STATE_FIPS.csv

```
State Abbreviation,FIPS Code,State Name
AK,02,ALASKA
AL,01,ALABAMA
```

np.genfromtxt: flexible way to read columns

Example file: STATE_FIPS.csv

```
State Abbreviation,FIPS Code,State Name
AK,02,ALASKA
AL,01,ALABAMA
```

```
import numpy as np
infilename = 'STATE_FIPS.csv'
indat = np.genfromtxt(infilename,delimiter=',',dtype=None,names=True)
```

`np.genfromtxt`: flexible way to read columns

Example file: STATE_FIPS.csv

```
State Abbreviation,FIPS Code,State Name
AK,02,ALASKA
AL,01,ALABAMA
```

```
import numpy as np
infilename = 'STATE_FIPS.csv'
indat = np.genfromtxt(infilename,delimiter=',',dtype=None,names=True)
```

`delimiter=','` delimiter can be *anything*

`dtype=None` Numpy interprets column data types. If unknown, makes it a string

`names=True` Each column gets a data type and a name

`np.genfromtxt`: flexible way to read columns

Example file: STATE_FIPS.csv

```
State Abbreviation,FIPS Code,State Name
AK,02,ALASKA
AL,01,ALABAMA
```

```
import numpy as np
infilename = 'STATE_FIPS.csv'
indat = np.genfromtxt(infilename,delimiter=',',dtype=None,names=True)
```

`delimiter=','` delimiter can be *anything*

`dtype=None` Numpy interprets column data types. If unknown, makes it a string

`names=True` Each column gets a data type and a name

```
In [10]: indat
Out[7]:
array([('AK', 2, 'ALASKA'), ('AL', 1, 'ALABAMA'), ('AR', 5, 'ARKANSAS'), ...
dtype=[('State_Abbreviation', '|S2'), ('FIPS_Code', '<i4'), ('State_Name', '|S20')])
```

`np.genfromtxt`: flexible way to read columns

Example file: STATE_FIPS.csv

```
State Abbreviation,FIPS Code,State Name
AK,02,ALASKA
AL,01,ALABAMA
```

```
import numpy as np
infilename = 'STATE_FIPS.csv'
indat = np.genfromtxt(infilename,delimiter=',',dtype=None,names=True)
```

`delimiter=','` delimiter can be *anything*

`dtype=None` Numpy interprets column data types. If unknown, makes it a string

`names=True` Each column gets a data type and a name

```
In [10]: indat
Out[7]:
array([('AK', 2, 'ALASKA'), ('AL', 1, 'ALABAMA'), ('AR', 5, 'ARKANSAS'), ...
dtype=[('State_Abbreviation', '|S2'), ('FIPS_Code', '<i4'), ('State_Name', '|S20')])
```

N.B.→underscore replaces space in names