Research Computing with Python
Lecture 5: File Input and Output

Ramses van Zon

SciNet HPC Consortium

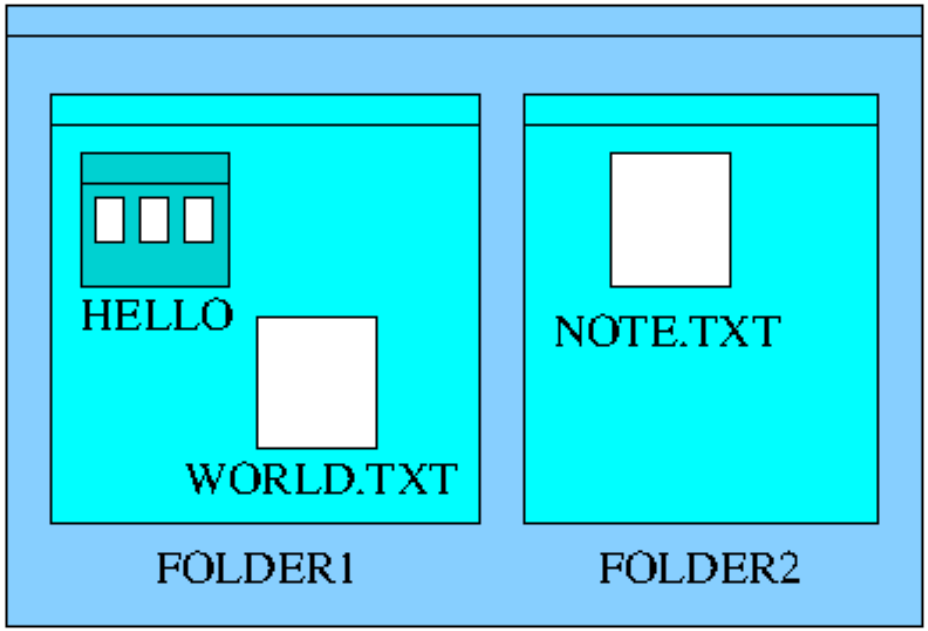
November 19, 2013

Today's Lecture

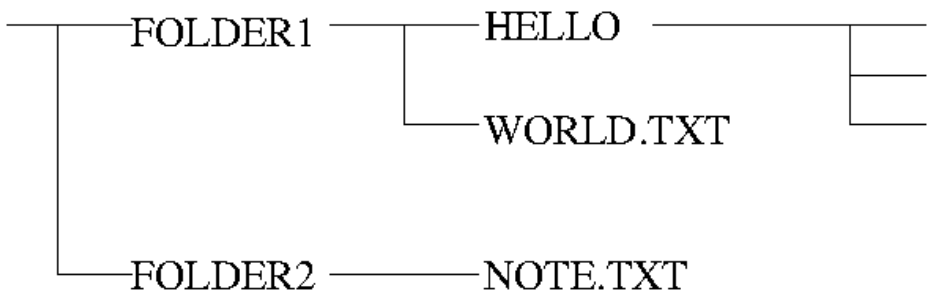
- Basic File Input and Output in Python
- Bit of file system theory, iops
- Different file formats (and how to use them)

SciNet compute • calcul CANADA

Basic File Input and Output in Python



Tree:



- Files contain your data
- Files are organized in directories or folders
- A directory is a file too
- Path: sequence of directories to get to a file

Files:

```
FOLDER1/WORLD.TXT
FOLDER2/NOTE.TXT
FOLDER1/HELLO/...
```

SciNet compute • calcul CANADA

Directories

Create

```
In [1]: import os
In [2]: os.mkdir('FOLDER1')
```

Change current directory

```
In [3]: os.chdir('FOLDER1')
In [4]: os.chdir('..')
```

SciNet compute • calcul CANADA

Write to a file

```
In [5]: f=open('FOLDER1/WORLD.TXT','w')
In [6]: line="Hello\n"
In [7]: f.write(line)
In [8]: f.close()
```

Appending

```
In [9]: f=open('FOLDER1/WORLD.TXT','a')
In [10]: line="World\n"
In [11]: f.write(line)
In [12]: f.close()
```

SciNet compute • calcul CANADA

Read a file

```
In [13]: f=open('FOLDER1/WORLD.TXT','r')
In [14]: line=f.readline()
In [15]: print line
Hello
In [16]: f.close()
```

Read/Write

```
In [17]: f=open('FOLDER1/WORLD.TXT','r+')
In [18]: f.seek(1)
In [19]: f.write('a')
In [20]: line=f.readline()
In [21]: print line
Hallo
In [22]: f.close()
```

Let's take a step back: some theory

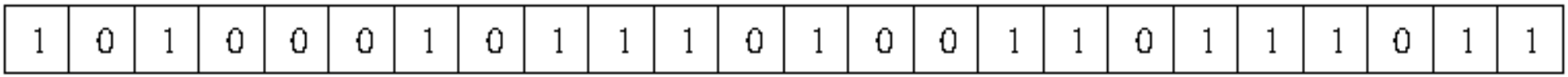
Computer Data Storage

Media:

- Memory
- Disks
- Flash (USB)
- DVD
- Tape
- ...

.

All media are essentially linear strings of bits:



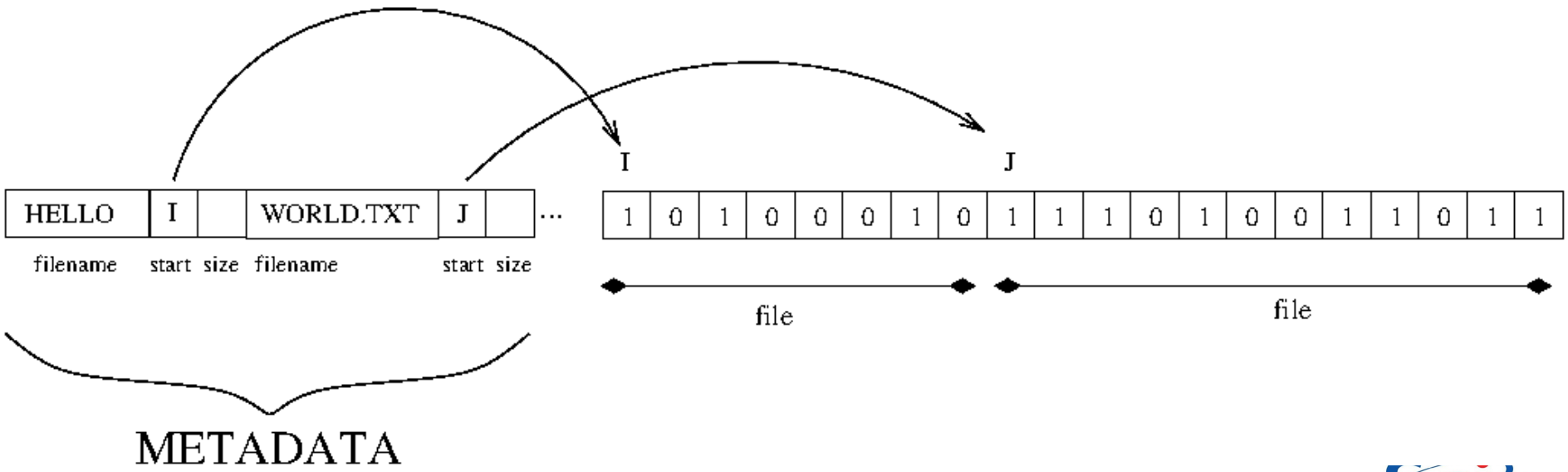
In and of itself, this is useless. What do these bits mean?

File systems

- Many non-volatile media use a file system
- This entails storing data describing the meaning of the data: **metadata**

Files

- Storage media is often subdivided into files
- Files have a name, a size and possibly other metadata
- Let's say that the metadata for the files is stored at the beginning of the storage media, e.g.



Metadata

Describes the file and its properties:

- File name
- File size
- Location on disk
- File type (though often through magic identifiers)
- Dates
- Read/write permissions
- ...

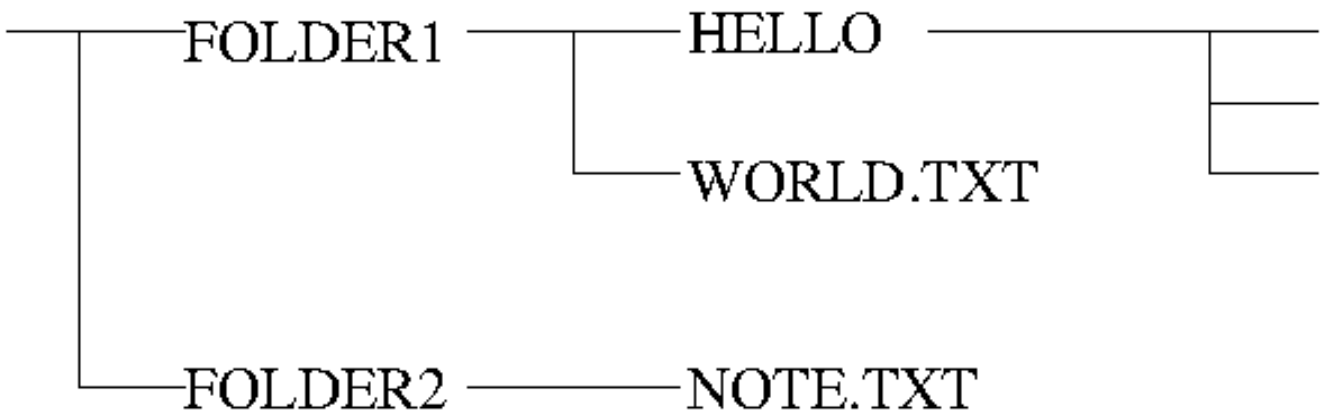
Directories or Folders

So we have files now, but this can get unorganized quickly. Imagine looking for the file 'NOTE.TXT' in a list of 10,000,000 files.

.

Directories

- Like special files that contain a list of (metadata for) other files.
- A directory can contain other directories, leading to a tree.



I/O Operations

What really happens if we open a file, write to it, etc.?

Opening a file

1. Find the file in the directory
   Or create a new entry in the directory
2. Check permissions on the file
3. Find the location of the file on disk
4. Initialize a file 'handle' and file 'pointer'
   The file handle is what `open` returns.

I/O Operations

What really happens if we open a file, write to it, etc.?

Writing to a file

1. Convert data to a stream of bytes.
2. Put those bytes in a buffer.
3. Update file pointer.
4. If buffer full: write to file

I/O Operations

What really happens if we open a file, write to it, etc.?

Reading from a file

1. If data not in buffer: read data into a buffer
2. Read bytes from buffer into variable, performing any needed conversion.
3. Update file pointer.

I/O Operations

What really happens if we open a file, write to it, etc.?

Closing a file

1. Ensure buffers are flushed to disk
2. Update any metadata.
3. Release buffers associated with the file handle.

Minimizing IOPS

- Disk I/O is usually the slowest part of a pipe line.
- If manipulating data from files is most of what you do, try and minimize iops.

Bad

```
s='Hi world\n'
for c in s:
  f=open('hiworld.txt','a')
  f.write(c)
  f.close()
```

Good

```
s='Hi world\n'

f=open('hiworld.txt','w')
f.write(s)
f.close()
```

- **Work in memory and reuse data if you can.**

SciNet compute • calcul CANADA

What's in a file?

Text

- Seems attractive: you can just read it.
- This is not as trivial as it may sound.
- Must assign a bit pattern to each letter or symbol (encoding).
- Ideally unique assignment across languages.

Binary

- Covered format of individual numbers in Numerics class.
- Decent binary format include information on the data in it, e.g.: hdf5. NetCDF.

SciNet compute • calcul CANADA

Text format

- ASCII Encoding: 7 bits = character
- 128 possible, but only 95 printable characters
- Uses 8-bit bytes: storage efficiency 82% at best.
- ASCII representation of floating point numbers:
  - Needs about 18 bytes vs 8 bytes in binary: **inefficient**
  - Representation must be computed: **slow**
  - **Non-exact** representation

ASCII

integers	characters
32	(space)
33-47	!"#$%&'()*+,-./
48-57	0–9
58-64	:;<=>?@
65-90	A–Z
91-96	[\]^_
97-122	a–z
123-126	{|}~

Text Encodings

ASCII: 7 bit encoding. For English.

Latin-1: 8 bit encoding. For western European Languages mostly.

UTF-8: *Variable-width* encoding that can represent every character in the Unicode character set.

Unicode: standard containing more than 110,000 characters.

Python can deal with these encodings:

```
# -*- coding: latin-1 -*-
print u"Comment ça va?"
```

Binary output

- Output the numbers as they are stored in memory
- Why bother: Fast and space-efficient.

Writing 128M doubles:

SciNet file system:	
ASCII	173 s
binary	6 s

ramdisk	
ASCII	174 s
binary	1 s

- Not human readable.
  But is that really so bad? If you have 100 million numbers in a file, are you going to read them all?

Why you should not use raw binary data

Just dumping the memory is fast, but you loose the information on what it meant. E.g.:

- Dump a 2d array of 100x100 floating point numbers
- Gives a file of 800,000 bytes.
- If we give this to someone else, how do they know what it is?
  - 2d array of 100x100 numbers
  - array of 10,000 floating point numbers,
  - string of 800,000 characters,
  - ...?

Binary Formats

You could invent your own binary format, but it's better to take an existing standard: Saves you potential bugs, the burden of documentation and/or maintaining an IO library, as one probably already exists.

Pickle: A python specific format. Portable for the same version.

NumPy: Has a binary format called `npy` or `npz`.

NetCDF: A self-describing format: contains not only data but names, descriptions of arrays (`scipy.io.netcdf`).

Hdf5: Another standard, self-describing format (`pytables`) Almost a filesystem in a file.

For both NetCDF and Hdf5, there are tools to inspect/analyze the files. Won't discuss Hdf5 here.

Pickle

- Base64 encoding using readable ASCII
- Portable for the same version of python.
- In the `pickle` module.
- Flexible, can `serialize` any structure.

```
In [23]: import pickle, os
In [24]: a=zeros((10000,10000))
In [25]: f=open('a.pickle','w')
In [26]: pickle.dump(a,f)
In [27]: close(f)
In [28]: print os.path.getsize('a.pickle')
3200000198
In [29]: g=open('a.pickle','r')
In [30]: b=pickle.load(g)
In [31]: g.close()
```

pickle.dump wall time: 121.44 s

SciNet compute • calcul CANADA

NumPy I/O Routines

- Remember shape
- Straight binary dump of data
- Surprisingly simple format but not ported too much.
- Just for NumPy arrays

```
In [32]: import os
In [33]: a=zeros((10000,10000))
In [34]: save('a.npy',a)
In [35]: print os.path.getsize('a.npy')
799997952
In [36]: b=load('a.npy')
```

numpy.save wall time: 1.21 s

Numpy I/O Routines

`save(FILE,ARRAY)` save a NumPy array to a .npy file

`savez(FILE,NAME1=ARRAY1,NAME2=ARRAY2)` save several NumPy arrays to an uncompressed zipped file with extension .npz

`savez_compressed(FILE,NAME1=ARRAY1,NAME2=ARRAY2)` save several NumPy arrays to a compressed zipped file with extension .npz

`load(FILE)` load NumPy array(s) from .npy (.npz) file. If FILE is an .npz, a dictionary with keys equal to the names supplied to savez is returned.

NetCDF files

There are three sections to a NetCDF file

Dimensions	How many points in each direction of our multidimensional array?
Variables	The data in our multidimensional array
Attributes	Variable and other annotations (e.g. units)

Python modules

- scipy.io.netcdf: for netcdf3 files
- netCDF4 (available in Canopy): for netcdf4 files

SciNet compute • calcul CANADA

NetCDF example

Can check the 'header' of an netcdf file using the linux utility `ncdump`:

```
$ ncdump -h test.nc
netcdf test {
dimensions:
    x = 1000 ;
variables:
    double a(x, x) ;
        a:units = "Kelvin" ;

// global attributes:
        :history = "This is a test" ;
}
```

Let's see how to create and use this file with scipy.io.netcdf.

scipy.io.netcdf: write file

```
In [37]: from scipy.io.netcdf import *

In [38]: f=netcdf_file('test.nc','w')       #create file

In [39]: f.history='This is a test' #set file attribute

In [40]: f.createDimension('x', 1000) #create dimension

In [41]: a=f.createVariable('a', 'd', ('x','x')) #array
In [42]: a[:]=zeros((1000,1000))                  #fill

In [43]: a.units='Kelvin'                  #array attribute

In [44]: f.close()                  #close file. Important!
```

scipy.io.netcdf: read file

```
In [45]: from scipy.io.netcdf import *
In [46]: f=netcdf_file('test.nc','r')
In [47]: print f.history
Created for a test

In [48]: a=f.variables['a']
In [49]: print a[100,300], a.units
0.0 Kelvin

In [50]: f.close()
```

scipy.io.netcdf overview

`HANDLE=netcdf_file(FILENAME,MODE)` Opens a netcdf file. `MODE='w'`for writing, `'r'` for reading, `MODE='rw'` for both.

`HANDLE.ATTRIBUTE=VALUE` Sets a file `ATTRIBUTE` to the value `VALUE`

`HANDLE.createDimension(NAME,VALUE)` Sets the dimension `NAME` (a string) to `VALUE`

`HANDLE.createVariable(NAME,SHAPE)` Creates the variabe `NAME` with `SHAPE` (a tuple of strings that were assigned a value with `createDimension`)

`HANDLE.variables[NAME]` The array variable `NAME`

`HANDLE.variables[NAME].ATTRIBUTE=VALUE` Set an attribute `ATTIBUTE` of the array variable `NAME` to the value `VALUE`

`HANDLE.close()` Flush everything to disk and close the file.

Final Tips

- If your data is not text, do not save it as text.
- Choose a binary format that is portable.
- Minimize IOPS: write/read big chunks at a time, don't `seek` more than needed, try to reuse data or load more in memory.
- Don't create millions of files: unworkable and slows down directories.
- Stick to letters, numbers, underscores and periods in file names.

Next Time

Next Lecture

Thursday November 21, 2013, 11:00 am

Topic: Visualization

SciNet compute • calcul CANADA