Exercise 1: Reading ASCII data files with Python

Aim: Explore reading a simple data file with functions and classes

Issues covered:

- Reading data from an ASCII text file
- Reading the header
- Using functions to read the file contents
- Using a class to read the file contents
- Defining the `__init__` initialisation method.
- Using `self` in class and method definitions.

1. Let's write "readHeader" and "readData" functions to read a simple CSV file.

a. Take a look at the contents of the "`example_data/weather_meta.csv`" file.
b. Open a new python module in an editor called "`read_weather.py`".
c. Write a "`read_header`" function that returns a dictionary of content found in the file header (the first 3 lines).
d. Test your "`read_header`" function with the "`example_data/weather_meta.csv`" file. Print the header dictionary.
e. Write a "`read_data`" function that returns a dictionary where each value is: `{column header: list of values}`.
f. Test the "`read_data`" function by printing the response.

2. Let's move the "readHeader" and "readData" functions in to a class.

a. Copy your "`read_weather.py`" module to a new module called "`read_weather_class.py`".
b. Create a `class` called "`Weather`" using: `class Weather(object):`
c. Modify the "`read_header`" and "`read_data`" functions so that they are methods of this class.
d. Within "`read_header`" assign the result to the instance variable "`self.header`".
e. Within "`read_data`" assign the result to the instance variable "`self.data`".
f. Add an "`__init__`" method that gets called when a class instance is created. This method should take in the file name parameter "`fname`".
g. Within the "`__init__`" method, call both the "`read_header`" and "`read_data`" methods.
h. Test the class by creating an instance and then printing the content:

```
weather = Weather("example_data/weather_meta.csv")
print(weather.header)
print(weather.data)
```

Solution 1: Reading ASCII data files with Python

1.

```
$ cat example_data/weather_meta.csv   # to look at the file
```

```
# Your "readHeader" function might look like:

def readHeader(fname):
    "Reads a data file `fname` and returns a dictionary of header metadata."
    with open(fname) as f:
        header = {}

        i = 0
        while i < 3:
            line = f.readline()
            # Strip any white space from line
            line = line.strip()

            key, value = line.split(":")
            header[key] = value
            i += 1

    return header

# Test it with
head = readHeader("example_data/weather_meta.csv")
print(head)
```

1. (continued…)

```
# Your "readData" function might look like:

def readData(fname):
    "Reads a data file `fname` and returns a dictionary of arrays of values."
    with open(fname) as f:
        data = {}

        # Ignore the header
        for i in range(3):
            f.readline()

        # Read in variable names
        col_names = f.readline().strip().split(",")
        for col_name in col_names:
            data[col_name] = []
            i = 0

        for line in f.readlines():
            # Strip any white space from line
            line = line.strip()

            values = line.split(",")

            for (i, value) in enumerate(values):
                col_name = col_names[i]
                data[col_name].append(value)

    return data

print(readData("example_data/weather_meta.csv"))
```

2.

```
$ cp read_weather.py read_weather_class.py
# Your new class might look like:
class Weather(object):

    def __init__(self, fname):
        self.readHeader(fname)
        self.readData(fname)

    def readHeader(self, fname):
        "Reads header from data file `fname` and populates dictionary: self.header."
        with open(fname) as f:
            self.header = {}

            i = 0
            while i < 3:
                line = f.readline()
                # Strip any white space from line
                line = line.strip()

                key, value = line.split(":")
                self.header[key] = value
                i += 1

    def readData(self, fname):
        "Reads a data file `fname` and populates instance dictionary: self.data."
        with open(fname) as f:
            self.data = {}

            # Ignore the header
            for i in range(3):
                f.readline()

            # Read in variable names
            col_names = f.readline().strip().split(",")
            for col_name in col_names:
                self.data[col_name] = []
                i = 0

            for line in f.readlines():

                # Strip any white space from line
                line = line.strip()
                values = line.split(",")

                for (i, value) in enumerate(values):
                    col_name = col_names[i]
                    self.data[col_name].append(value)

# Test it with:
weather = Weather("example_data/weather_meta.csv")
print(weather.header)
print(weather.data)
```

Exercise 2: Basic NetCDF command-line tools

Aim: Introduce the use of ncdump, ncgen and CDL to manipulate NetCDF

Issues covered:

- The CDL language as a readable ASCII form of NetCDF
- Working with ncdump to display the contents of a NetCDF file
- Creating a NetCDF file directly from CDL using ncgen

1. Let's look at the contents of an existing NetCDF file with ncdump.

a. Locate the file "`example_data/tas_rcp45_2055_ann_avg_change.nc`" – this contains a future projection of change in annual surface temperature for the 2050s.
b. Look at the header with:
```
ncdump -h example_data/tas_rcp45_2055_ann_avg_change.nc
```
c. Try piping the above command in to "`less`" so that you can view page by page.
d. Type "`ncdump`" on its own to view the command line options.
e. Use ncdump command-line options to view the type (a.k.a. "kind") of NetCDF file.
f. Use ncdump command-line options to view the values of the "time" variable.
g. Use ncdump command-line options to view the values of the "time" variable as date-time strings.
h. Use ncdump and the `grep` utility to only view the `standard_name` attributes in the header of the file: "`example_data/ggas2014121200_00-18.nc`".

2. Let's write our own CDL file and convert it to NetCDF.

a. Open a new file called "`weather.cdl`" in an editor.
b. Open the data file "`example_data/weather.csv`" in a separate editor window.
c. Write the first line of the CDL file: "`netcdf weather {`"
d. Define one dimension of length 3: "`time`"
e. Define three variables against the "`time`" dimension: "`time`", "`temp`", "`rainfall`"
f. Define units for each variable as follows: "`days since 2014-01-01 00:00`", "`degrees_c`", "`mm`".
g. Define one global attribute: `comment = "Created as an exercise";`
h. Add the data for each variable in to the CDL file.
i. Run "`ncgen -o weather.nc weather.cdl`" to create your NetCDF file.
j. Try generating some Fortran code to write the NetCDF file using:
```
"ncgen -l f77 weather.cdl"
```

Solution 2: Basic NetCDF command-line tools

1.

```
$ ls example_data/tas_rcp45_2055_ann_avg_change.nc
example_data/tas_rcp45_2055_ann_avg_change.nc

$ ncdump -h example_data/tas_rcp45_2055_ann_avg_change.nc | less
netcdf tas_rcp45_2055_ann_avg_change {
dimensions:
        lon = 360 ;
...
$ ncdump -k example_data/tas_rcp45_2055_ann_avg_change.nc
classic
$ ncdump -v time example_data/tas_rcp45_2055_ann_avg_change.nc | grep "time ="
time = 74189.5 ;

$ ncdump -t -v time example_data/tas_rcp45_2055_ann_avg_change.nc | grep "time ="
time = "2065-12-30 12" ;

$ ncdump -h example_data/ggas2014121200_00-18.nc | grep standard_name
                longitude:standard_name = "longitude" ;
...
                V10:standard_name = "northward_wind" ;
```

2.

```
$ gedit weather.cdl &
$ gedit example_data/weather.csv &
...after lots of editing...

$ cat weather.cdl
netcdf weather {
dimensions:
    time = 3 ;
variables:
    float time(time) ;
        time:units = "days since 2014-01-01 00:00" ;
    float temp(time) ;
        temp:units = "degrees_c" ;
    float rainfall(time) ;
        rainfall:units = "mm" ;
// global attributes:
    :comment = "Created as an exercise" ;
data:
 time = 0, 0.5, 1 ;
 temp = 2.34, 6.7, -1.34 ;
 rainfall = 4.45, 8.34, 10.25 ;
}
$ ncgen -o weather.nc weather.cdl
$ ncgen -l f77 weather.cdl
```

Exercise 3: Reading NetCDF files with Python

Aim: Introduce reading a file using the netCDF4 Python library

Issues covered:

- Importing netCDF4
- Reading a NetCDF file as a Dataset
- Accessing Dimensions
- Accessing Variables
- Accessing global and Variable attributes

1. Let's work with the netCDF4 library to examine the contents of a data file.

a. Import the "`netCDF4`" library.
b. Open the file "`example_data/ggas2014121200_00-18.nc`" as a netCDF4 `Dataset`.
c. Loop through the variables in the `Dataset` listing their ids (which are the keys of the dictionary).
d. Assign the python variable "`sst`" to the NetCDF `Variable` in the file called "`SSTK`". Print the variable.
e. Loop through the dimensions of the variable and print their Id and length.
f. Print the shape and size of the "`sst`" variable.
g. Loop through the NetCDF attributes of "`sst`" and print them (use "`sst.ncattrs()`" to get their names).
h. Keep this data in your python session for the next example.

2. Let's extract some data and its related coordinate information and metadata.

a. Take a slice of "`sst`" as follows and assign it to the variable "`arr`":
```
[1, 0, 10:20, 30:35]
```
b. Find out what type of object "`arr`" is.
c. Assign a list of the "`sst`" dimensions to the variable "`dims`". Print this variable.
d. Assign the dictionary of `Dataset` variables to the variable "`vars`".

e. Now extract the slices from each `Dataset` variable matching those in "`arr`". Assign them to the following variables: `arr_time, arr_level, arr_lats, arr_lons.`
f. Loop through the four new variables and print their values.
g. Create an empty dictionary called "`metadata`". Loop through the NetCDF Variable attributes of "`sst`" and copy them into this new dictionary. Print the dictionary.
h. Keep this data in your python session for the next exercise.

Solution 3: Reading NetCDF files with Python

1.

```
$ python
>>> import netCDF4
>>> ds = netCDF4.Dataset("example_data/ggas2014121200_00-18.nc")
>>> for v in ds.variables:
...     print(v)

>>> sst = ds.variables["SSTK"]
>>> print(sst)

>>> for d in sst.dimensions:
...     print(d, len(ds.variables[d]))

>>> print(sst.shape, sst.size)

>>> for attr in sst.ncattrs():
...     print(attr, "=", getattr(sst, attr))
```

2.

```
#...continuing from exercise 1...
>>> arr = sst[1, 0, 10:20, 30:35]
>>> print(type(arr))

>>> dims = sst.dimensions
>>> print(dims)

>>> vars = ds.variables
>>> arr_time = vars["time"][1]
>>> arr_level = vars["surface"][0]
>>> arr_lats = vars["latitude"][10:20]
>>> arr_lons = vars["longitude"][30:35]

>>> for vals in (arr_time, arr_level, arr_lats, arr_lons):
...     print(vals)

>>> metadata = {}
>>> for attr in sst.ncattrs():
...     metadata[attr] = getattr(sst, attr)

>>> print(metadata)
```

Exercise 4: Writing NetCDF files with Python

Aim: Introduce writing a data file using the Python netCDF4 library

Issues covered:

- Importing netCDF4
- Defining Variables
- Defining Dimensions
- Defining global and Variable attributes
- Writing a NetCDF file as a Dataset

1. Let's write the data/metadata from the previous exercise to a new NetCDF file.

a. Import "`Dataset`" from the "`netCDF4`" library and "`numpy`" as "`np`".
b. Open a new netCDF4 `Dataset` for writing, to a file "`mydata.nc`", specify the write mode ("`w`") and the format ("`NETCDF4_CLASSIC`"). Assign to the variable "`myds`".
c. Create four `Dimensions` from your previous slices in the last exercise. Re-use the names from the last exercise. Note that the "`level`" and "`time`" `Dimensions` should have length 1.
d. Create four `Variables` from those dimensions and assign them following this example for times:
```
times = myds.createVariable('time', np.float64, ('time',))
```

e. Create "`myvar`" as a new `Dataset Variable`, with id "`temp`", type "`np.float64`" and dimensions: "`time`", "`level`", "`lat`", "`lon`".
f. Remove the "`_FillValue`" value in the "`metadata`" dictionary. The next step will not work unless we do this. Fill values should be handled when the `Variable` is created but we are ignoring that fact for this example.
g. Write `Variable` attributes from the key/value pairs found in the input data (held in the "`metadata`" dictionary).
h. Write one global attribute to the "`myds`" `Dataset`. Set the attribute "`source`" to the value "`super dataset`".

i. Write some data values to each of your spatiotemporal variables. Use simple lists of integers for these. Make sure they are the right length matching the slices from the last exercise.
j. Write the data from the previous exercise (called "`arr`") to your "`dset`" `Dataset` using:
```
myds[:] = arr
```
k. Close the `Dataset` "`myds`" to write the file.
l. Use `ncdump` at the linux command-line to view the file contents:
```
$ ncdump mydata.nc
```

Solution 4: Writing NetCDF files with Python

1.

```
$ python

>>> from netCDF4 import Dataset
>>> import numpy as np

>>> myds = Dataset('mydata.nc', 'w', format='NETCDF4_CLASSIC')

>>> time = myds.createDimension('time', 1)
>>> level = myds.createDimension('level', 1)
>>> lat = myds.createDimension('lat', 10)
>>> lon = myds.createDimension('lon', 5)

>>> times = myds.createVariable('time', np.float64, ('time',))
>>> levels = myds.createVariable('level', np.int32, ('level',))
>>> latitudes = myds.createVariable('latitude', np.float32, ('lat',))
>>> longitudes = myds.createVariable('longitude', np.float32, ('lon',))

>>> myvar = myds.createVariable('temp', np.float32,
                                  ('time','level','lat','lon'))

>>> del metadata["_FillValue"]

>>> for (key, value) in metadata.items():
       myvar.setncattr(key, value)
>>> myds.source = "super dataset"

>>> times[:] = arr_time
>>> levels[:] = arr_level
>>> latitudes[:] = arr_lats
>>> longitudes[:] = arr_lons

>>> myvar[:] = arr

>>> myds.close()
```

```
$ ncdump mydata.nc
```