pyDataView - a framework for data analysis

By Edward Smith

28th March 2017

Overview

- Introduction to Python
- Overview of pyDataView and motivation for its develoment
- Outline of a general design philosophy for data anaylsis
- Demonstration using pyDataView

Python

- Interpreted language: command prompt or running scripts.py
- A range of data types, everything is an object.

```
a = 3.141592653589          # Float
i = 3                       # Integer
s = "some string"           # String
l = [1,2.54,"hello"]        # List
d = {"red":4, "blue":5}     # Dictonary
x = np.array([1,2,3])       # Numpy array
```

- Usual constructs including conditionals (if statements) iterators (for loops) and functions (def name) with space defining scope
- Use external libraries numpy and matplotlib for scientific computing

Example of Functions

```
#Define a variable
a = 5.0
```

Comment

Function name

Tell Python you are defining a function

```
#Define Function
def square(input):
    "calculate square"
    output = input*input
    return output
```

Name of input variable to the function

Document function here "text" for one line or """ multi-line verbose and descriptive text """

Level of indent determines what is inside the function definition. Variables defined (scope) exists only inside function. Ideally 4 spaces and avoid tabs. See PEP 8

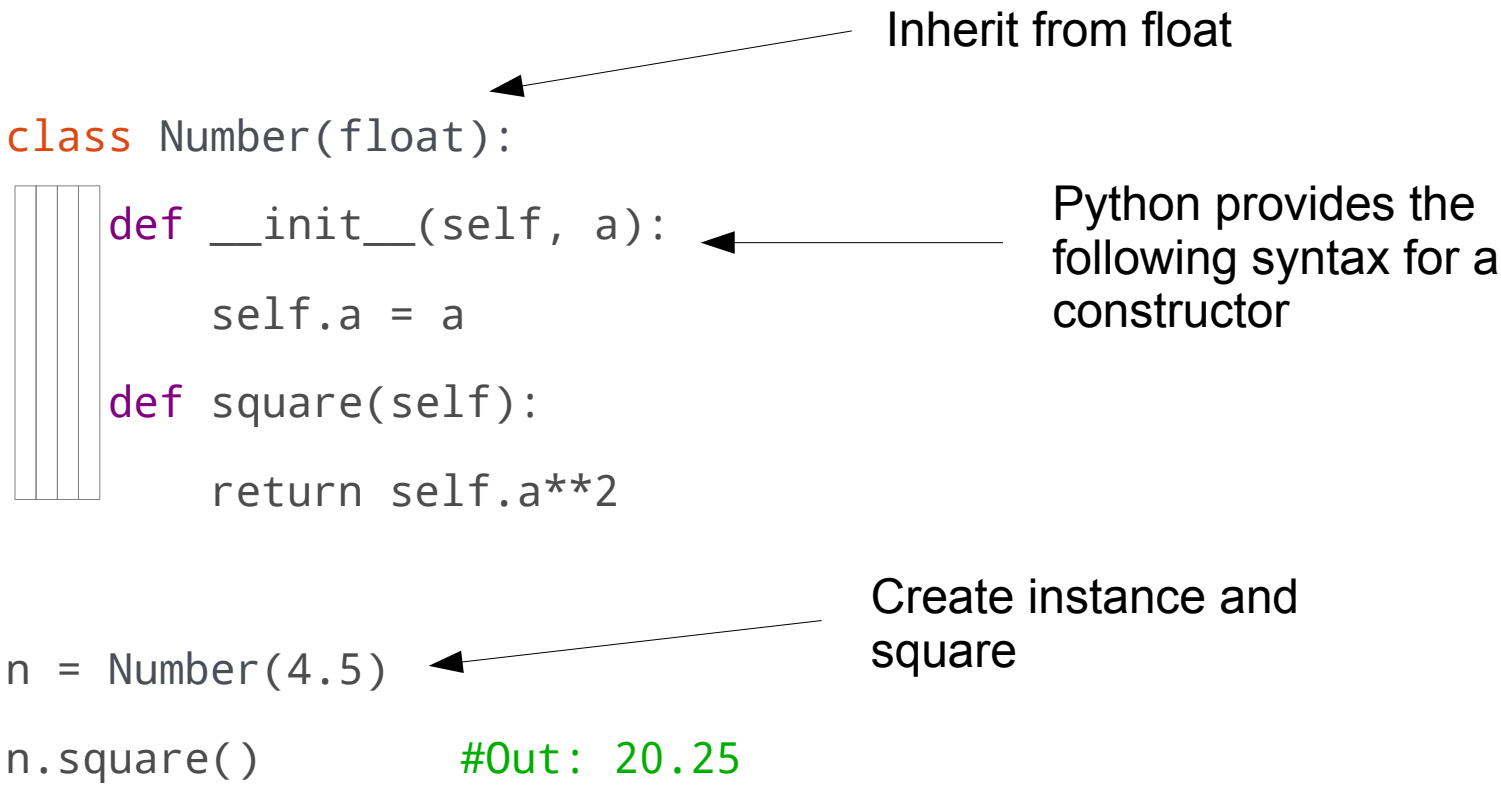
Operation on input variable

Value to return from function

```
#We call the function like this
square(a) Out: 25.0
```

Classes in Python

- A number class which includes constructor and method to get square



Why Python – Strings

- String manipulations

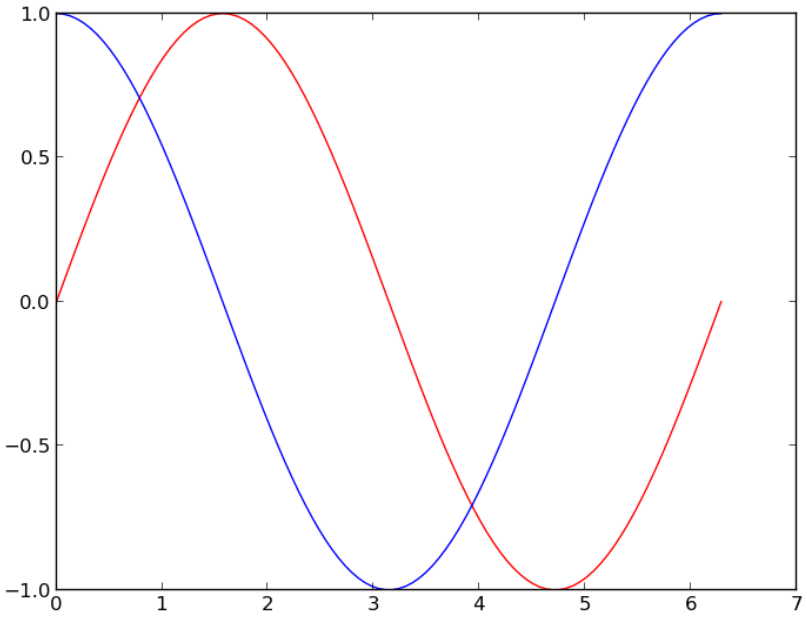
```
s = "some string"
t = s + " with more"     Out: "some string with more"
s*3   Out: "some stringsome stringsome string"
s[3]                              Out: e
s[0:4]                            Out: some
s.capitalize()                    Out: "Some string"
s.split(" ")                      Out: ["some", "string"]
s.find("o")                       Out: 1
t = s.replace("some", "a")        Out: t="a string"
```

- In ipython, use tab to check what functions (methods) are avaliable

Python add-on modules Numpy/Matplotlib

```
#python
import numpy as np
import matplotlib.pyplot as plt


x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)
z = np.cos(x)
plt.plot(x, y)
plt.plot(x, z)
plt.show()
```



A GUI with a Slider

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.widgets as mw

#Setup initial plot of sine function
x = np.linspace(0, 2*np.pi, 200)
l, = plt.plot(x, np.sin(x))

#Adjust figure to make room for slider
plt.subplots_adjust(bottom=0.15)
axslide = plt.axes([0.15, 0.05, 0.75, 0.03])
s = mw.Slider(axslide, 'A value', 0., 5.)

#Define function
def update(A):
    l.set_ydata(np.sin(A*x))
    plt.draw()

#Bind update function to change in slider
s.on_changed(update)
plt.show()
```

Adjust figure to make room for the slider and add a new axis axslide for the slider to go on

Define a function to change figure based on slider value. Here this updates the plot data and redraws the plot

Bind function update to slider change

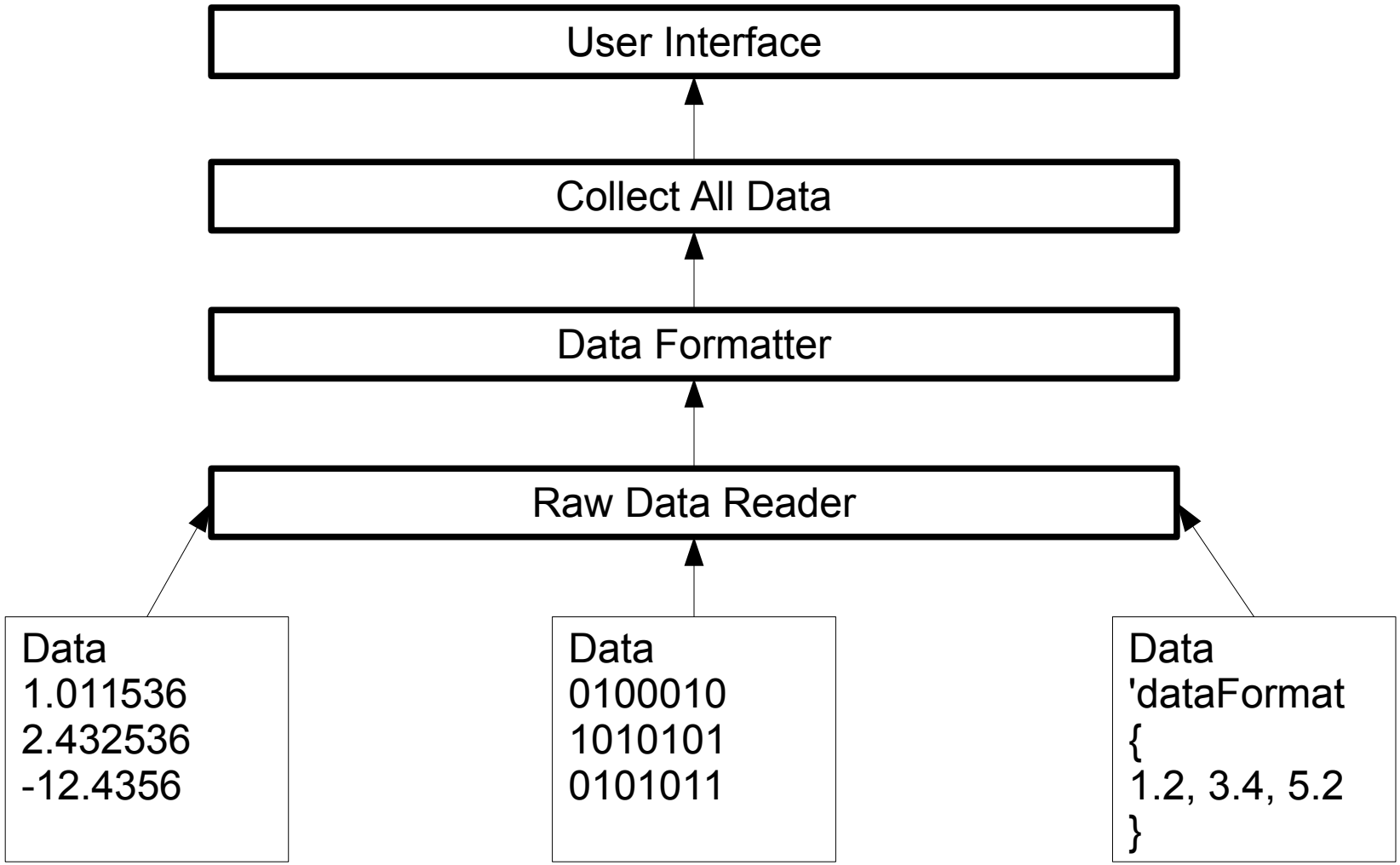
A Typical Postprocessing Workflow

- Get data from some source: experiments, numerical simulation, surveys/studies, an internet database, big data, etc.
- Import it into python as a single numpy array, a list of numpy arrays, a dictonary of values, etc.
- Play around with various plots and data analysis techniques.
- Take the most promising output and save the script which generates this exactly, add labels and format to publication quality.
- We can develop an automated process from data to figure with minimimal user input. This is useful because
  - Easy to make changes when required by reviewers
  - Clearer mapping from data to output (Opendata movement)
  - Create functions to break the analysis down and reduce errors
  - You can use the same scripts to analyse similar data

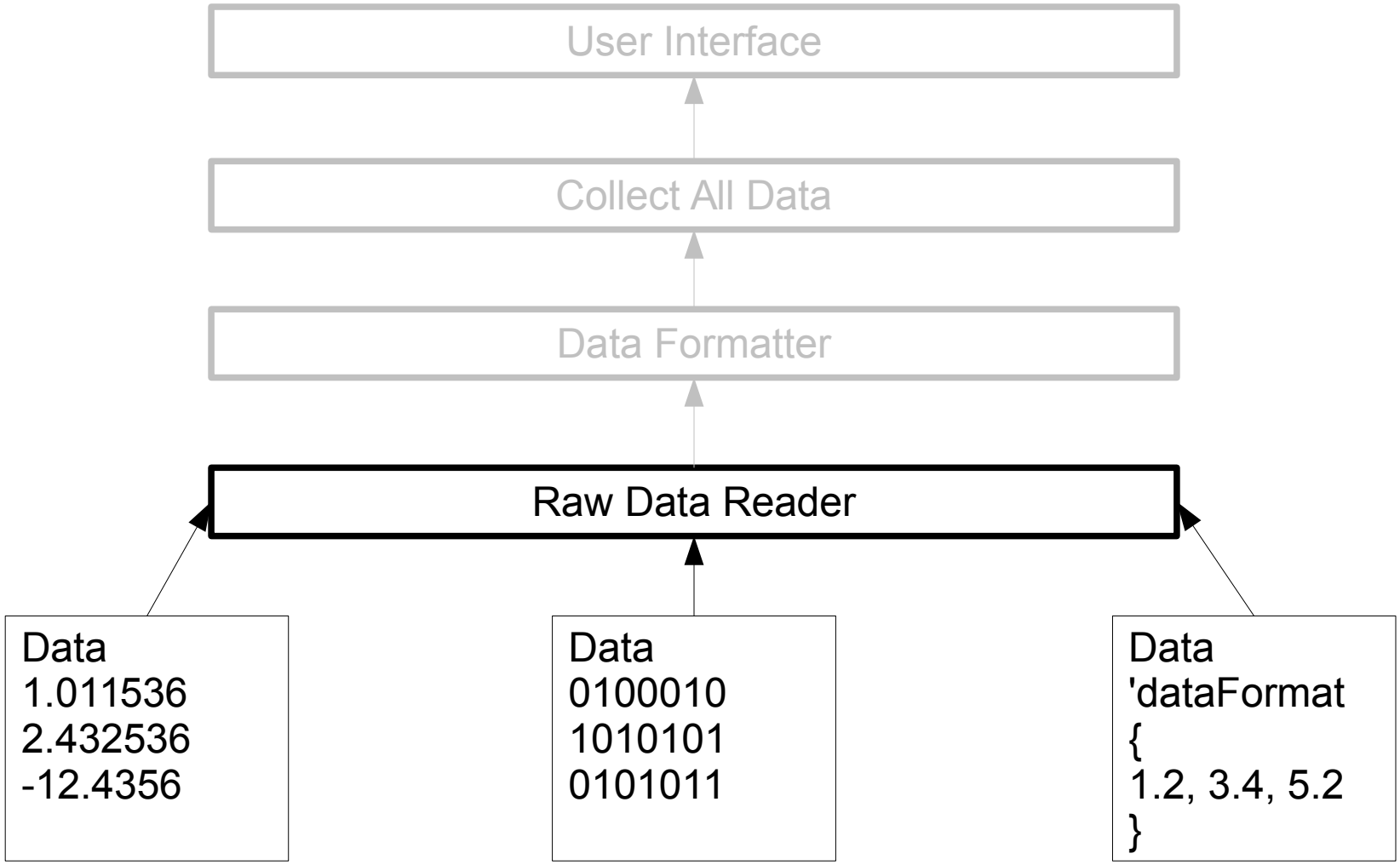
pyDataView

- Evolved from lots of data analysis on large data sets (Tb split over many files of Mb size)
- Inspired by ease of use of sliceomatic on MATLAB for quick exploration of mulit-dimensional data
- Open-source and fairly small code base with emphesis on writing plug-in code for new cases
- Generate script to make publication figures or go beyond basic capability
- https://github.com/edwardsmith999/pyDataView

pyDataView


User Interface
Collect All Data
Data Formatter
Raw Data Reader
Data
1.011536
2.432536
-12.4356
Data
0100010
1010101
0101011
Data
'dataFormat
{
1.2, 3.4, 5.2
}

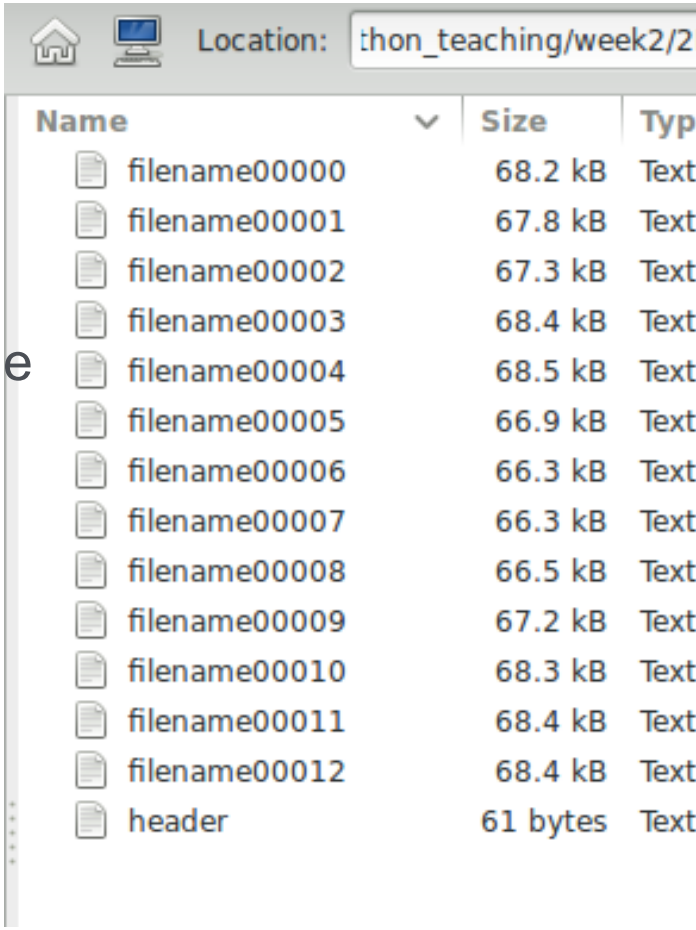
pyDataView


User Interface
Collect All Data
Data Formatter
Raw Data Reader
Data
1.011536
2.432536
-12.4356
Data
0100010
1010101
0101011
Data
'dataFormat
{
1.2, 3.4, 5.2
}

Raw Data Reader

- Use the format method, with prepended zeros, so files are displayed in order in folder (and read in order):

```
for i in range(100):
    print("filename{:05}".format(i))
```



- Get contents of all folder containing filename

```
import glob
for i in glob.glob("filename*"):
    print(i)
```

- **We want to TAKE FOLDER and RETURN LIST OF FILES**

Raw Data Reader

1) Reading data stored as an ascii csv file

```
data = np.genfromtxt(filename)
```

```
1.025468
2.0198
-3.2471
...
```

2) Reading data stored as a binary file

```
data = np.fromfile(open(filename,'rb'), dtype='d')
```

```
01000101
01010101
01011010
...
```

3) Reading data stored as text

```
f = open(filename)

filestr = f.read()

idx=filestr.find('FoamFile')

d = filestr[idx:].split('\n')[2:5]
```

```
/*---------------------
| \      /  F ield
|  \    /   O peration
|   \  /    A nd
|    \/     M anipulation
\*-----------------------
FoamFile
{
    Nx        84;
    Nz        50;
    Lx        1560.
```

Raw Data Reader

4) Reading open-source HDF5 format (large binary data, self documenting) using python package h5py

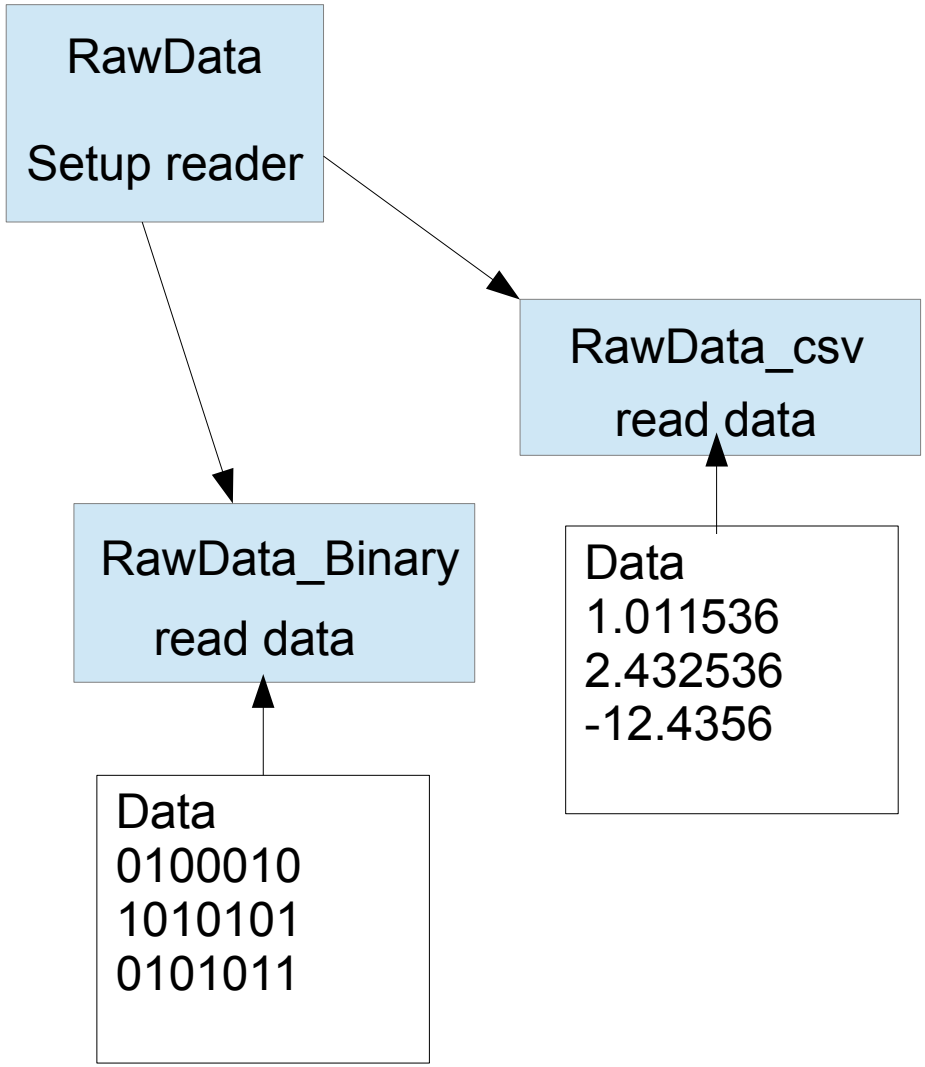
```
import h5py
f = h5py.File(fpath,'r')
data = f[u'data'].items()[0][1]
```

5) Another common format is vtk, open-source for 3D graphics visualization but I've had limited success reading: packages like vtk, pyvtk, mayavi/TVTK,

```
import vtk
reader = vtk.vtkUnstructuredGridReader()
reader.SetFileName(filename)
reader.ReadAllVectorsOn()
reader.ReadAllScalarsOn()
reader.Update()
```

- **We want to TAKE FILENAME and RETURN DATA**

Raw Data Reader



The base class defines the constructor, but does not specify how to read

Inherit and define read for each data type

Raw Data Reader

```
class RawData():
        ...
    def read(self, startrec, endrec):
        raise NotImplemented
class RawData_binary(RawData):
    def read(self, startrec, endrec):
        for f in self.files[startrec:endrec]:
            data[:,:,:,:,i] = np.fromfile(open(f,'rb'), dtype='d')
class RawData_csv(RawData):
    def read(self, startrec, endrec):
        for f in self.files[startrec:endrec]:
            data[:,:,:,:,i] = np.genfromtxt(f)
```

The base class defines the constructor, but does not specify how to read

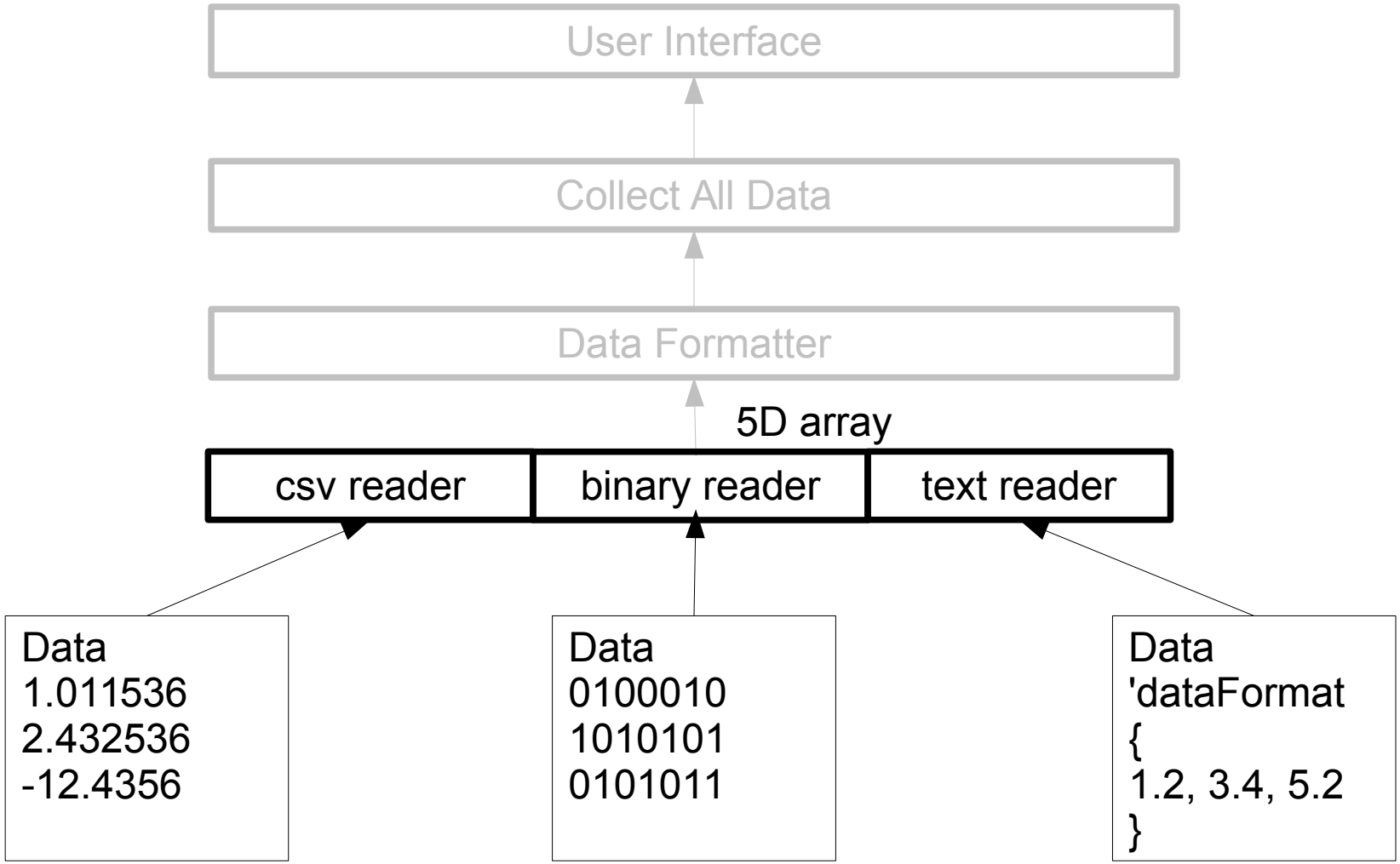
Inherit and define read for each data type

Return a 5D array [nx, ny, nz,nperbin, nrecs] with nrecs=endrec-startrec+1 and nperbin is 1 for scalar field, 3 for vector field..

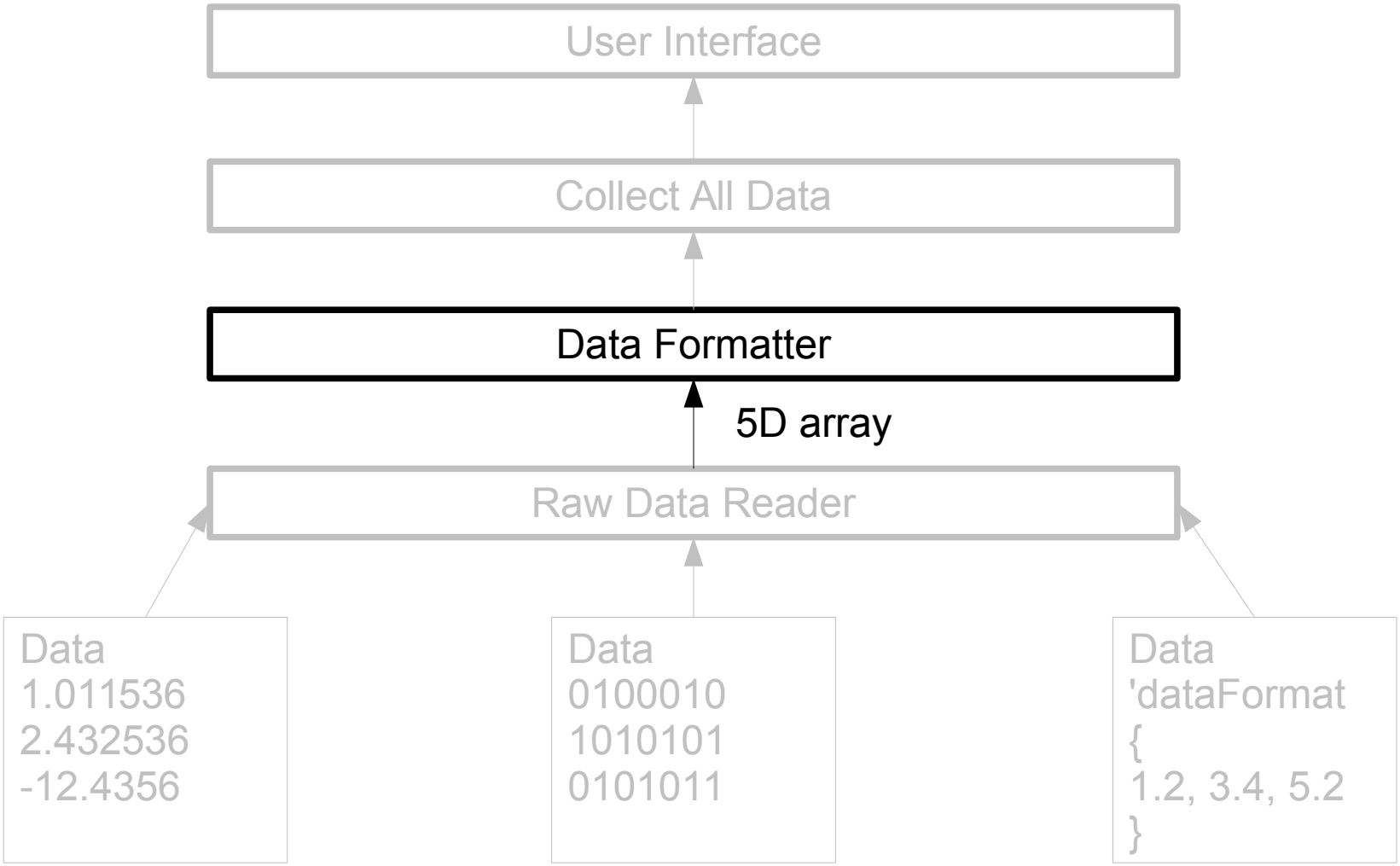
The Function Interface

- The inputs to a function and returned output are like a contract with the user, 'give me this and I will give you that'
  - From all data we aim to returned the same data from different file formats. This could include an interface to internet data
  - This means the same top level code is used for any format
  - This hides the form of the underlying data from the user, you only need to call read(filename) to get the data
- When releasing software, version number systems are based around interface
  - From v1.0 to v1.1 the interface stays the same
  - If major number changes, e.g. v1.1 to v2.0, the interface has changed and is no longer backward compatible

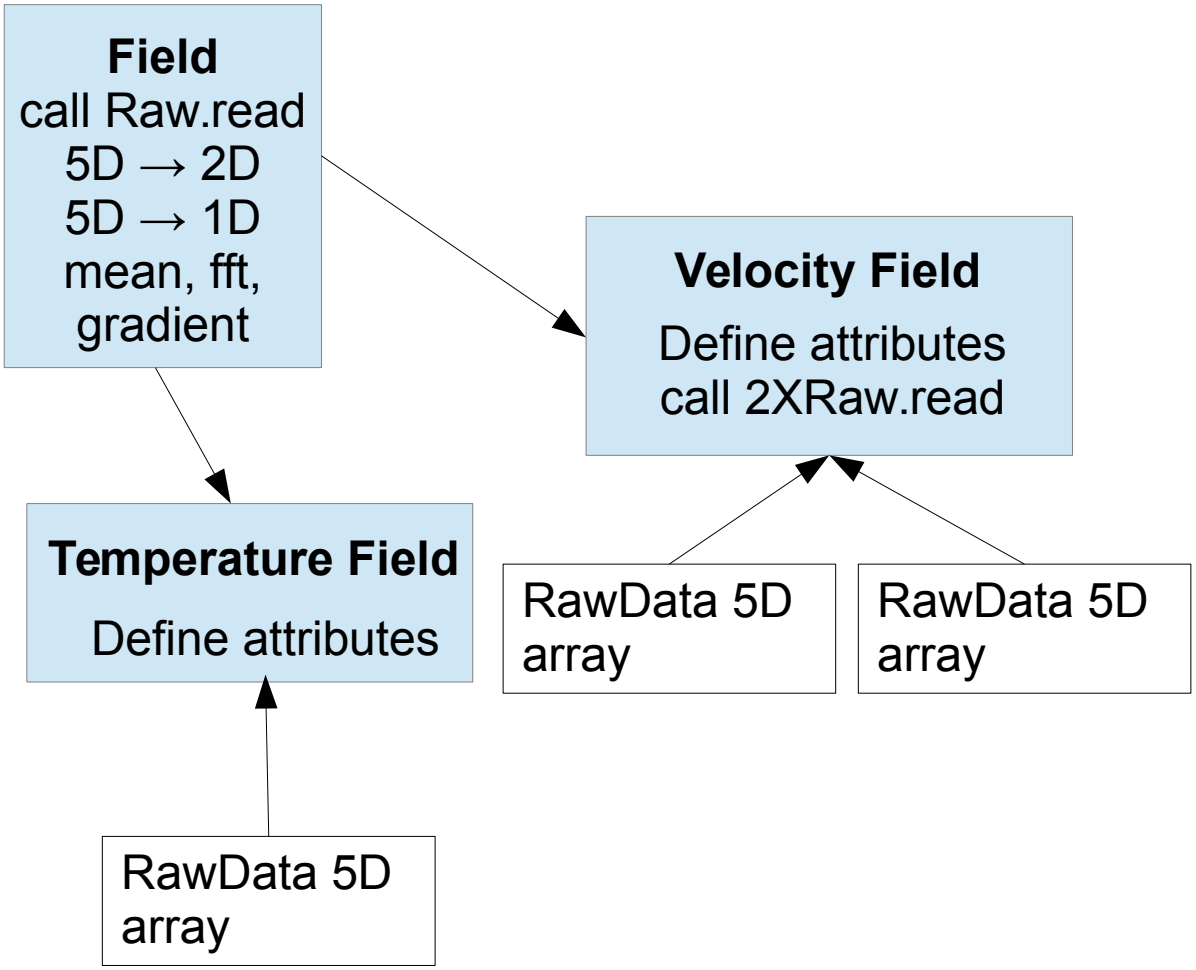
pyDataView


User Interface
Collect All Data
Data Formatter
5D array
csv reader
binary reader
text reader
Data
1.011536
2.432536
-12.4356
Data
0100010
1010101
0101011
Data
'dataFormat
{
1.2, 3.4, 5.2
}

pyDataView


User Interface
Collect All Data
Data Formatter
5D array
Raw Data Reader
Data
1.011536
2.432536
-12.4356
Data
0100010
1010101
0101011
Data
'dataFormat
{
1.2, 3.4, 5.2
}

Data Formatter – Some Field



Data Formatter – Some Field

- We can now plot any type of data but we need to format for ouput

```
class Field():
    def __init__(self, fdir):
        self.Raw = RawData_binary(fdir, self.fname,
                                  self.dtype, self.nperbin)
        ...
    def read(startrec,endrec):
       return self.Raw.read(startrec,endrec)
    def contour(startrec, endrec):
        f = self.read(startrec, endrec)
        return np.mean(f[:,:,:,0,:],(0,3))
    def profile(startrec, endrec):
        ...
    def fft # Return the Fourier Transform, etc
    def gradient # Return the gradient
```

Take the mean to get a 2D array from 5D data.

Take the mean to get a 1D array from 5D data.

Data Formatter – Some Field

- We can now plot any type of data but we need to format for ouput

```
class Temperature(Field):
    dtype = 'd'
    nperbin = 1
    fname = 'Tdata'
```

The Field class is designed to be associated with a type of data

Data Formatter – Velocity Field

- We can combine multiple raw data types in one field

```
class VelocityField(Field):
    dtype = 'd'
    nperbin = 3
    fname = 'u'
    def __init__(self, fdir):
        mObj = RawData_binary(fdir, 'mass', self.dtype, 1)
        muObj = RawData_binary(fdir, 'momentum', self.dtype, 3)
    def read(startrec, endrec):
        m = mObj.read(startrec, endrec)
        mu = muObj.read(startrec, endrec)
        u = mu/m
        return u
```

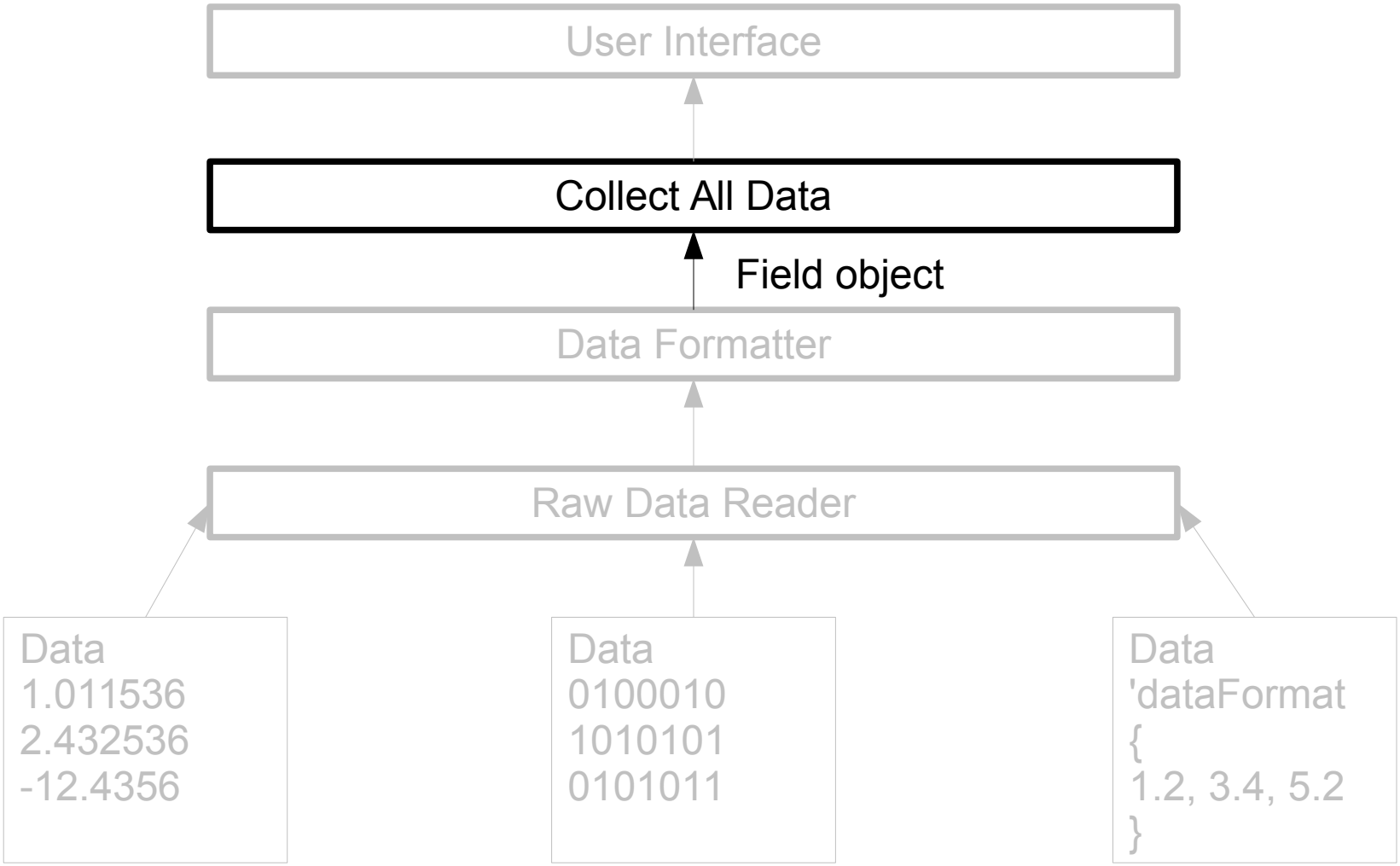
Velocity is momentum over mass

$$\mathbf{u} = \frac{\mathbf{mu}}{\mathrm{m}}$$

Data Formatter – Key Points

- We define a new data formatter inhereting from Field to ensure the same interface and a range of data analysis tools
- User only defines attributes which in the simplest case determines the call to the raw data reader
- More complex cases combine the data from different files in some way and returns the combined data as a 5D array
- Field base class provides helper functions to reduce 5D data to 1D line plots, 2D contour plots, 3D isosurfaces or analyse data using Fourier transforms, gradients, mean values, etc.

pyDataView


User Interface
Collect All Data
Field object
Data Formatter
Raw Data Reader
Data
1.011536
2.432536
-12.4356
Data
0100010
1010101
0101011
Data
'dataFormat
{
1.2, 3.4, 5.2
}

Collect All Data

- Look for all data in the formats you want

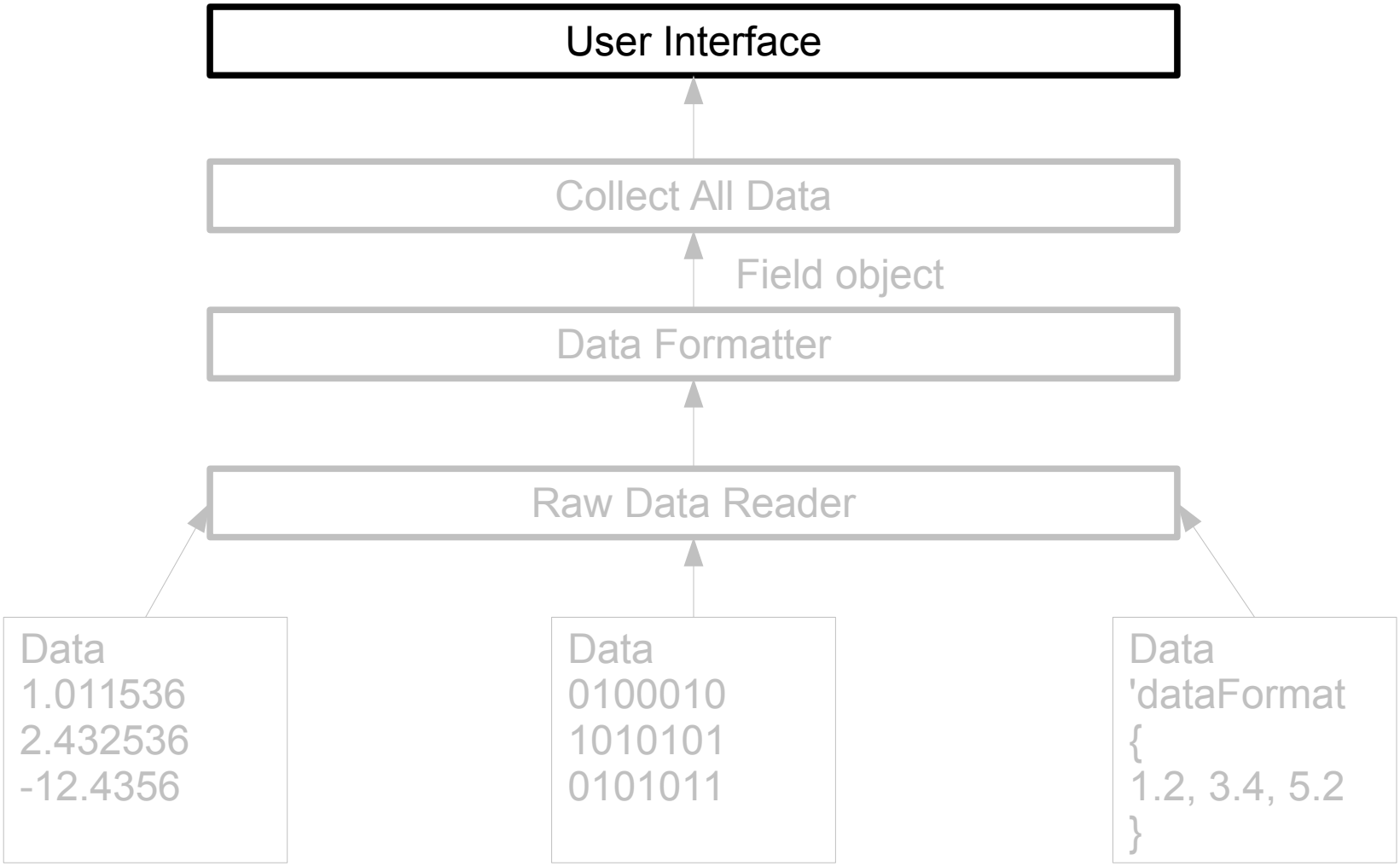
```
class Some_PostProc(PostProc):
    def __init__(self,resultsdir,**kwargs):
        self.resultsdir = resultsdir
        #List all fields you would like to look for in resultsdir
        possibles = {'Temperature': Temperature,
                     'Velocity': VelocityField}

        self.plotlist = {}
        for key, field in possibles.items():
            try:
                self.plotlist[key] = field(self.resultsdir)
            except AssertionError:
                pass
```

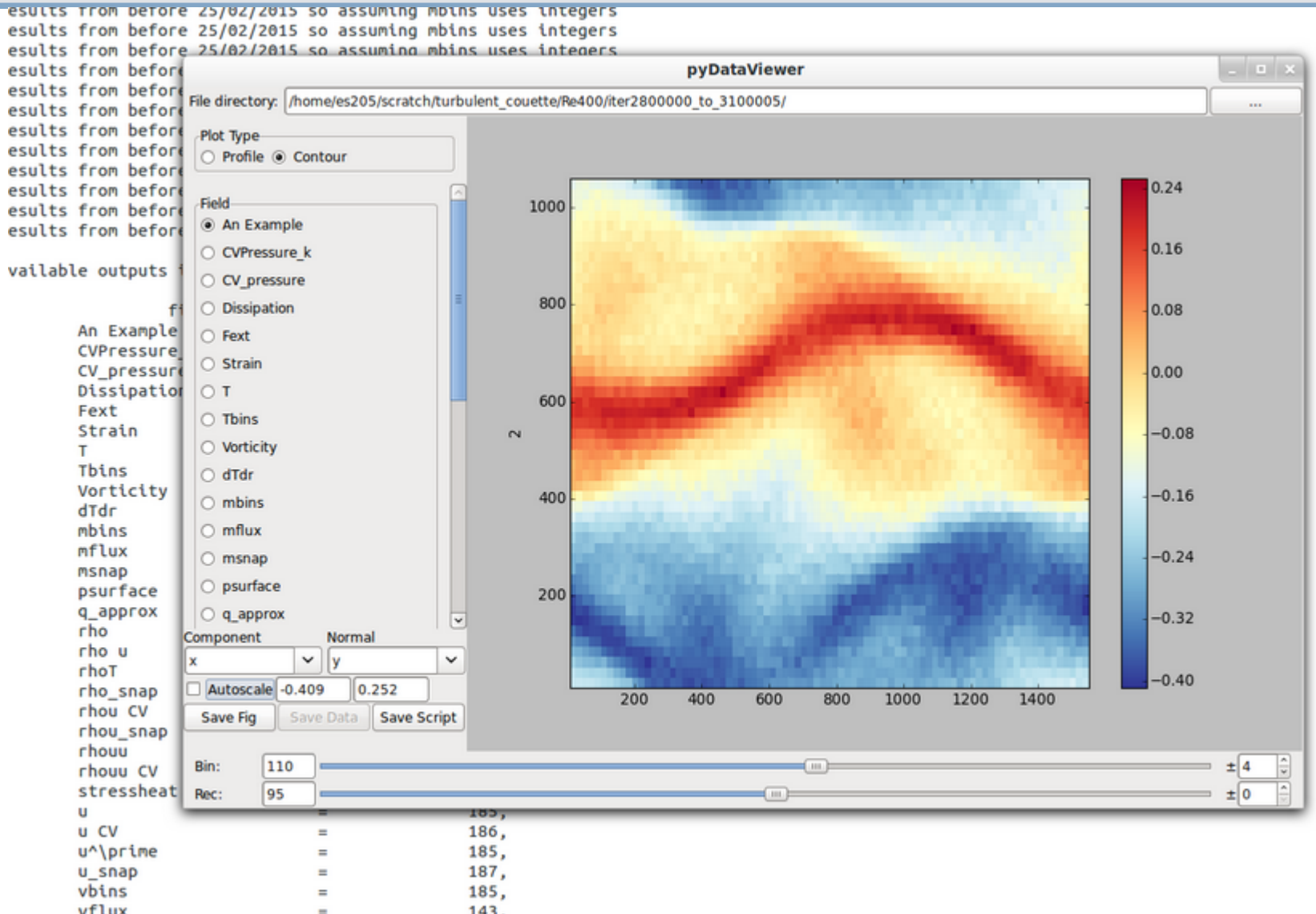
Collect All Data – Key Points

- We define a new data collector, inhereting from Postproc, which asks which data types we want to look for.
- Just a Dictonary of human readable names and the fieldtypes which they correspond to.
- This then tries to load these data formats one by one
- Contains a range of print and display functions to check which data is available.
- All possible fields are contained in `plotlist` and can be loaded as needed.

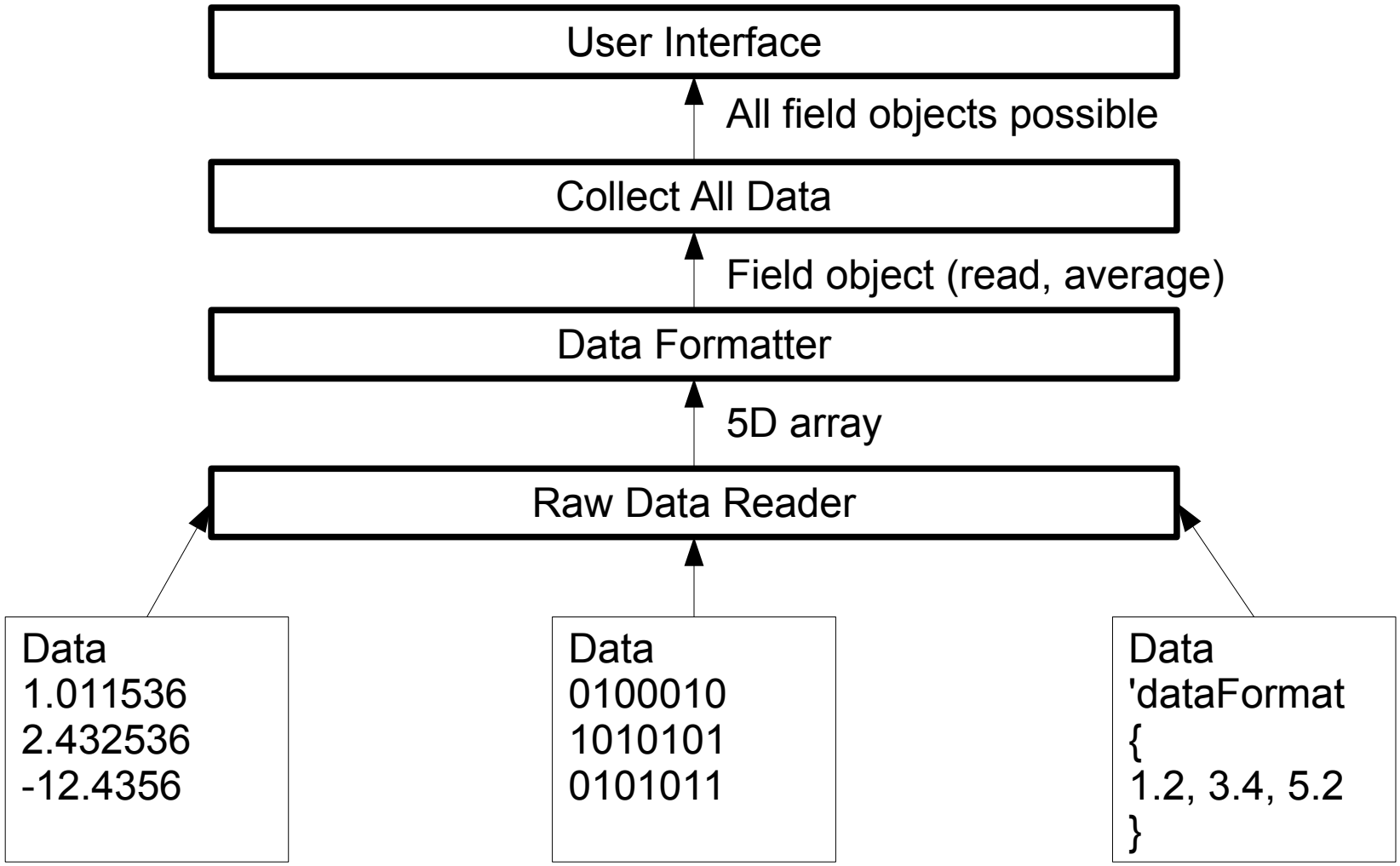
pyDataView


User Interface
Collect All Data
Field object
Data Formatter
Raw Data Reader
Data
1.011536
2.432536
-12.4356
Data
0100010
1010101
0101011
Data
'dataFormat
{
1.2, 3.4, 5.2
}

wxPython Interface to Allow Quick Exploration of Data



pyDataView


User Interface
All field objects possible
Collect All Data
Field object (read, average)
Data Formatter
5D array
Raw Data Reader
Data
1.011536
2.432536
-12.4356
Data
0100010
1010101
0101011
Data
'dataFormat
{
1.2, 3.4, 5.2
}

Summary

- pyDataView is open-source and freely available with the GNU v3 license. Collaborators welcome!
- More important is the general design philosophy for your own data anaylsis:
  - Design an interface to your raw data (or collector) to return a single format (here 5D arrays)
  - Another layer uses one or more raw data sources, to create useful fields of data and provides methods to manipulate these data fields.
  - The final layer(s) collect all possible forms of data together and provides easy access, for example through a graphical user interface (GUI)