Table Management in Python

by Kaustubh Vaghmare

(IUCAA, Pune)

E-mail: kaustubh[at]iucaa[dot]ernet[dot]in

What we shall cover?

If we chose to stay behind by an year or more, the following modules.

- asciitable
- atpy

But today, we shall cover the "table" sub-module inside Astropy.

”atpy” and ”asciitable” are no longer developed.

They have been absorbed by the astropy core package.

But you must still have them installed.

- Some codes you are given may be based on them.
- Some modules may require them.

But while learning, you must learn the astropy versions namely

- astropy.io.ascii
- astropy.table

astropy.io.ascii vs. astropy.table

- astropy.io.ascii is meant purely for reading and writing tables.
- Is a collection of "extensible" classes which can be extended to support newer formats.

astropy.table

- builds upon io.ascii using its functionality for reading / writing tables
- and adding its own powerful table operations.

You won't need to read much about io.ascii unless your tables have some special outstanding features.

In Brief - The ”Class” Concept

We have discussed the concept of an "object" earlier.

- Objects have well defined behavior.
- They have methods which help you perform supported operations on them.
- Where are all these rules defined?

A "class" is crudely put, a definition which allows one to create objects.

To create table objects, we will need a Table class.

Let's Start

In [1]:

```
# First we need the Table class to create table objects.
# The warning that will be flashed has so far not affected
# any functional features of Table class
from astropy.table import Table
```

```
/usr/local/lib/python2.7/dist-packages/IPython/zmq/__init__.py:
65: RuntimeWarning: libzmq 4 detected.
        It is unlikely that IPython's zmq code will work proper
ly.
        Please install libzmq stable, which is 2.1.x or 2.2.x
  RuntimeWarning)
```

In [2]:

```
# Next we need to create the Table object using a file.
demo_table = Table.read("demo.txt", format = "ascii")
```

What if the table does not load?

If you get errors when using read() method, it means that your file is formatted in a way that the standard parser is unable to understand the structure of your file.

What to do? Understand the io.ascii.read() method in detail and supply the various options to Table.read().

eg. `header_start = ";"` or `delimiter="|"` ,etc.

Displaying Tables.

In [3]:

```
print demo_table
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
 M31 2012-01-02  17.0  17.5
 M31 2012-01-02  17.1  17.4
M101 2012-01-02  15.1  13.5
 M82 2012-02-14  16.2  14.5
 M31 2012-02-14  16.9  17.3
 M82 2012-02-14  15.2  15.5
M101 2012-02-14  15.0  13.6
 M82 2012-03-26  15.7  16.5
M101 2012-03-26  15.1  13.5
M101 2012-03-26  14.8  14.3
```

In [4]:

```
demo_table.pprint() # Does exactly the same thing.
# but you can supply options such as
# max_lines, max_width, show_unit, show_name
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
 M31 2012-01-02  17.0  17.5
 M31 2012-01-02  17.1  17.4
M101 2012-01-02  15.1  13.5
 M82 2012-02-14  16.2  14.5
 M31 2012-02-14  16.9  17.3
 M82 2012-02-14  15.2  15.5
M101 2012-02-14  15.0  13.6
 M82 2012-03-26  15.7  16.5
M101 2012-03-26  15.1  13.5
M101 2012-03-26  14.8  14.3
```

In [5]:

```
# In this example, we are suppressing column names from appearin
g.
demo_table.pprint(show_name=False)
```

```
 M31 2012-01-02 17.0 17.5
 M31 2012-01-02 17.1 17.4
M101 2012-01-02 15.1 13.5
 M82 2012-02-14 16.2 14.5
 M31 2012-02-14 16.9 17.3
 M82 2012-02-14 15.2 15.5
M101 2012-02-14 15.0 13.6
 M82 2012-03-26 15.7 16.5
M101 2012-03-26 15.1 13.5
M101 2012-03-26 14.8 14.3
```

More Ways to Print Tables.

Using an interactive table scrolling tool.

```
demo_table.more()
```

Or display it as a formatted table in a browser.

```
demo_table.show_in_browser()
```

Quickly Check Basic Properties of Loaded Table

In [6]:

```
print len(demo_table) # Number of rows.
```

```
10
```

In [10]:

```
print demo_table.colnames # The names of the columns.
```

```
['name', 'obs_date', 'mag_b', 'mag_v']
```

You can also print any meta information, if available.

```
demo_table.meta
```

Accessing Columns of the Table

In [11]:

```
print demo_table["name"] # one column
```

```
name
----
 M31
 M31
M101
 M82
 M31
 M82
M101
 M82
M101
M101
```

In [12]:

```
print demo_table["name", "mag_b"] # more than one column
```

```
name mag_b
---- -----
 M31  17.0
 M31  17.1
M101  15.1
 M82  16.2
 M31  16.9
 M82  15.2
M101  15.0
 M82  15.7
M101  15.1
M101  14.8
```

Accessing Rows in a Table

In [13]:

```
print demo_table[0] # SADLY, row objects do not support printing.
```

```
<Row 0 of table
 values=('M31', '2012-01-02', 17.0, 17.5)
 dtype=[('name', 'S4'), ('obs_date', 'S10'), ('mag_b', '<f8'),
('mag_v', '<f8')]>
```

In [14]:

```
demo_table[0].data # is one way to get values in a row.
```

Out[14]:

```
('M31', '2012-01-02', 17.0, 17.5)
```

In [17]:

```
lines = demo_table.pformat() # a list of strings, each string a row, includes header.
print lines[2]
```

```
 M31 2012-01-02  17.0  17.5
```

Individual Element Access

```
In [18]: demo_table["name"][0]
```

```
Out[18]: 'M31'
```

```
In [19]: demo_table[0]["name"] # also works the same as above.
```

```
Out[19]: 'M31'
```

Sub-sectioning Tables

In [20]:
```
subsection_col = demo_table["name","mag_b"] # by column.
```

In [21]:
```
subsection_row = demo_table[2:5] # by rows.
```

In [22]:
```
subsection_row2 = demo_table[ [1,5,3] ]
```

In [23]:
```
subsection_both = demo_table["name", "mag_b"] [1:5]
```

Changing elements inside a Table

- You know how to access columns, rows and individual elements.
- Using = sign, you can assign the selected col, row or element another value.

So,

```
demo_table["name"] = ... list of 10 names
demo_table["name"] = "SingleName"
```

will both work.

In [24]:

```
print demo_table
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
 M31 2012-01-02  17.0  17.5
 M31 2012-01-02  17.1  17.4
M101 2012-01-02  15.1  13.5
 M82 2012-02-14  16.2  14.5
 M31 2012-02-14  16.9  17.3
 M82 2012-02-14  15.2  15.5
M101 2012-02-14  15.0  13.6
 M82 2012-03-26  15.7  16.5
M101 2012-03-26  15.1  13.5
M101 2012-03-26  14.8  14.3
```

```
In [25]: demo_table["name"] = "X"
         print demo_table
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
   X 2012-01-02  17.0  17.5
   X 2012-01-02  17.1  17.4
   X 2012-01-02  15.1  13.5
   X 2012-02-14  16.2  14.5
   X 2012-02-14  16.9  17.3
   X 2012-02-14  15.2  15.5
   X 2012-02-14  15.0  13.6
   X 2012-03-26  15.7  16.5
   X 2012-03-26  15.1  13.5
   X 2012-03-26  14.8  14.3
```

Adding New Columns

In [26]:

```
# Method 1
demo_table["NewColumn"] = range(len(demo_table))
print demo_table
```

```
name  obs_date  mag_b mag_v NewColumn
---- ---------- ----- ----- ---------
   X 2012-01-02  17.0  17.5         0
   X 2012-01-02  17.1  17.4         1
   X 2012-01-02  15.1  13.5         2
   X 2012-02-14  16.2  14.5         3
   X 2012-02-14  16.9  17.3         4
   X 2012-02-14  15.2  15.5         5
   X 2012-02-14  15.0  13.6         6
   X 2012-03-26  15.7  16.5         7
   X 2012-03-26  15.1  13.5         8
   X 2012-03-26  14.8  14.3         9
```

In [30]:

```
# Method 2, using Column Object
from astropy.table import Column
newcol = Column( data = range(len(demo_table)), name = "NewColN"
)
demo_table.add_column( newcol, index = 0)
print demo_table
```

```
NewColN name  obs_date  mag_b mag_v NewColumn
------- ---- ---------- ----- ----- ---------
      0    X 2012-01-02  17.0  17.5         0
      1    X 2012-01-02  17.1  17.4         1
      2    X 2012-01-02  15.1  13.5         2
      3    X 2012-02-14  16.2  14.5         3
      4    X 2012-02-14  16.9  17.3         4
      5    X 2012-02-14  15.2  15.5         5
      6    X 2012-02-14  15.0  13.6         6
      7    X 2012-03-26  15.7  16.5         7
      8    X 2012-03-26  15.1  13.5         8
      9    X 2012-03-26  14.8  14.3         9
```

Removing Columns

In [32]:

```
demo_table.remove_columns(["NewColN", "NewColumn"])
print demo_table
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
   X 2012-01-02  17.0  17.5
   X 2012-01-02  17.1  17.4
   X 2012-01-02  15.1  13.5
   X 2012-02-14  16.2  14.5
   X 2012-02-14  16.9  17.3
   X 2012-02-14  15.2  15.5
   X 2012-02-14  15.0  13.6
   X 2012-03-26  15.7  16.5
   X 2012-03-26  15.1  13.5
   X 2012-03-26  14.8  14.3
```

For Rows

Similar functions exist. Please read documentation for details. Or explore using iPython.

```
demo_table.remove_row(5)
demo_table.remove_rows( [5,6])
demo_table.remove_rows( slice(3,6) )
```

Table Sorting

In [33]:

```
demo_table = Table.read("demo.txt", format="ascii")
print demo_table
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
 M31 2012-01-02  17.0  17.5
 M31 2012-01-02  17.1  17.4
M101 2012-01-02  15.1  13.5
 M82 2012-02-14  16.2  14.5
 M31 2012-02-14  16.9  17.3
 M82 2012-02-14  15.2  15.5
M101 2012-02-14  15.0  13.6
 M82 2012-03-26  15.7  16.5
M101 2012-03-26  15.1  13.5
M101 2012-03-26  14.8  14.3
```

In [35]:

```
demo_table.sort(["name", "mag_b"]) # sort by name, then mag_b
```

```
In [36]: print demo_table
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
M101 2012-03-26  14.8  14.3
M101 2012-02-14  15.0  13.6
M101 2012-01-02  15.1  13.5
M101 2012-03-26  15.1  13.5
 M31 2012-02-14  16.9  17.3
 M31 2012-01-02  17.0  17.5
 M31 2012-01-02  17.1  17.4
 M82 2012-02-14  15.2  15.5
 M82 2012-03-26  15.7  16.5
 M82 2012-02-14  16.2  14.5
```

In [37]:

```
demo_table.reverse() # Reverse existing table. Descending order!
print demo_table
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
 M82 2012-02-14  16.2  14.5
 M82 2012-03-26  15.7  16.5
 M82 2012-02-14  15.2  15.5
 M31 2012-01-02  17.1  17.4
 M31 2012-01-02  17.0  17.5
 M31 2012-02-14  16.9  17.3
M101 2012-03-26  15.1  13.5
M101 2012-01-02  15.1  13.5
M101 2012-02-14  15.0  13.6
M101 2012-03-26  14.8  14.3
```

Table Groups

- It is possible to organize the table into groups.
- For example, all entries for object M101 can be selected as a single group.
- One can access individual groups for various operations.
- Also supported "group-wise reductions"

In [40]:

```
demo_table = Table.read("demo.txt", format="ascii")
grouped_table = demo_table.group_by("name")
```

In [41]:

```
# To access groups.
print grouped_table.groups[0] # first group
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
M101 2012-01-02  15.1  13.5
M101 2012-02-14  15.0  13.6
M101 2012-03-26  15.1  13.5
M101 2012-03-26  14.8  14.3
```

Group-wise Reductions (eg. group-wise mean)

In [42]:

```
import numpy
grouped_table.groups.aggregate( numpy.mean)
```

```
WARNING:astropy:Cannot aggregate column 'obs_date'
```

```
WARNING: Cannot aggregate column 'obs_date' [astropy.table.grou
ps]
```

Out[42]:

name	mag_b	mag_v
M101	15.0	13.725
M31	17.0	17.4
M82	15.7	15.5

Filters

- Define a function `some_filter( TableObject, KeyColumns )` .
- The function return True or False.
- Then use the function to remove rows which satisfy some condition.

eg. write a filter to select rows whose mean is positive.

```
def positive_mean( table, key_colnames) :
if np.mean( table["ColName"] > 0:
    return True
else
    return False

t_positive_mean = t_grouped.groups.filter( positive_mean )
```

Stuff For You To Explore On Your Own

Stacks - vstack, hstack

”joins”