Analysis, Reporting and Visualization

Python-based Analysis Infrastructure

DataSource/DataSink Classes for Retrieval and Storage of LDMS Data

Extensible Transform Classes for Efficient, Pipelined Analysis

Grafana Dashboard Support

Python Infrastructure

Simply and easy to prototype new analysis

Comprehensive Numpy and SciPy support libraries

Data plane in C for performance

Control plane in Python for accelerated development

DataSource/DataSink Classes

Generic interface data that supports multiple storage formats

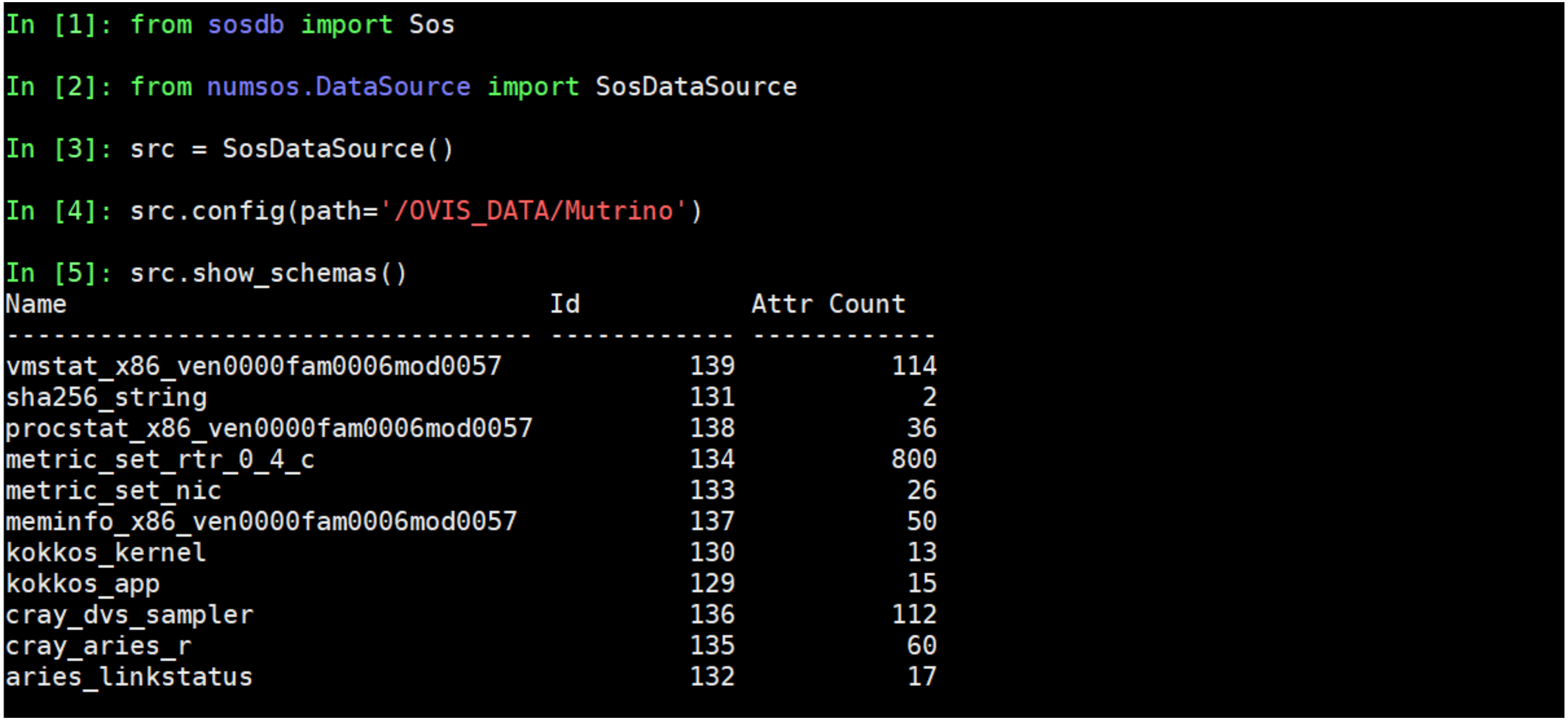
SQL *Familiar* API

Interfaces to Ease Debug/Development

DataSource Class

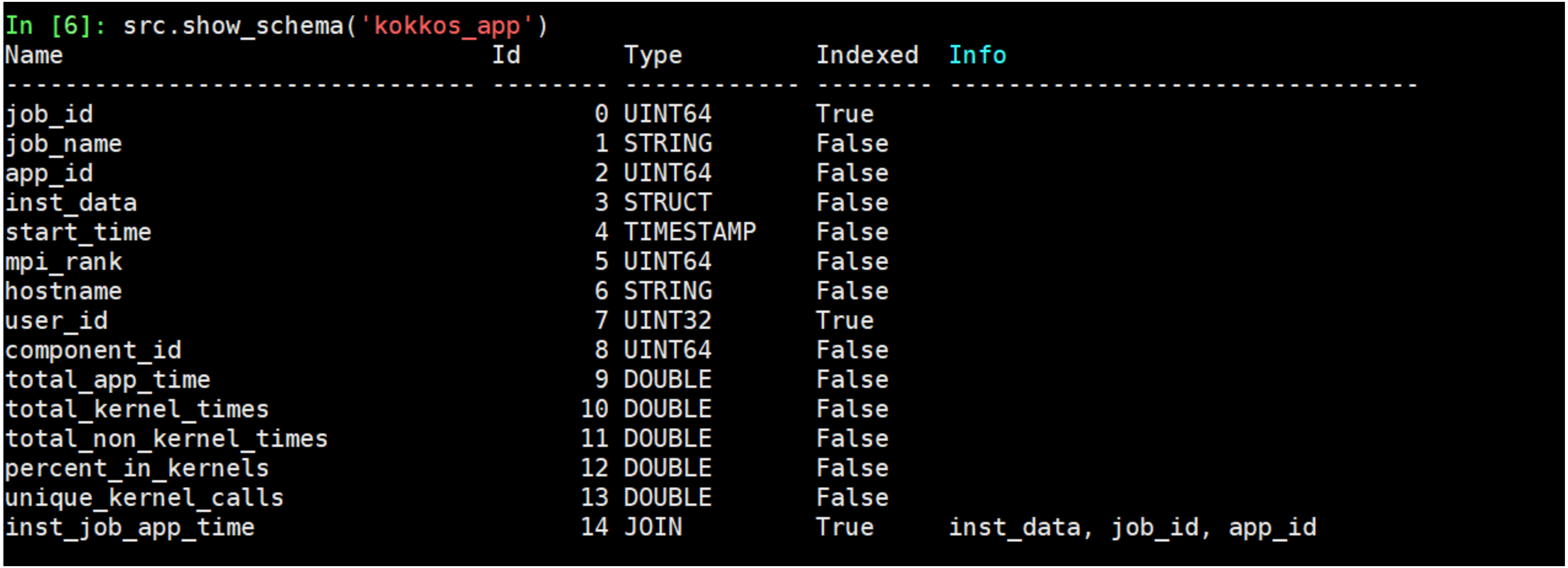
- Single base class to access various storage backends
- API designed to facilitate analysis that would work with CSV, SOS, or other data sources
- DataSource.config
  - Specify where and how data is to be accessed
  - Parameters may be specific to each *back-end*
- DataSource.select
  - SQL like syntax for identifying what, where and in what order the data is to be returned
- DataSource.show
  - Present the data for output
- DataSource.get_results
  - Return data for analysis

DataSource – open and configure



```
In [1]: from sosdb import Sos

In [2]: from numsos.DataSource import SosDataSource

In [3]: src = SosDataSource()

In [4]: src.config(path='/OVIS_DATA/Mutrino')

In [5]: src.show_schemas()
Name                               Id           Attr Count
---------------------------------- ------------ ------------
vmstat_x86_ven0000fam0006mod0057            139          114
sha256_string                               131            2
procstat_x86_ven0000fam0006mod0057          138           36
metric_set_rtr_0_4_c                        134          800
metric_set_nic                              133           26
meminfo_x86_ven0000fam0006mod0057           137           50
kokkos_kernel                               130           13
kokkos_app                                  129           15
cray_dvs_sampler                            136          112
cray_aries_r                                135           60
aries_linkstatus                            132           17
```

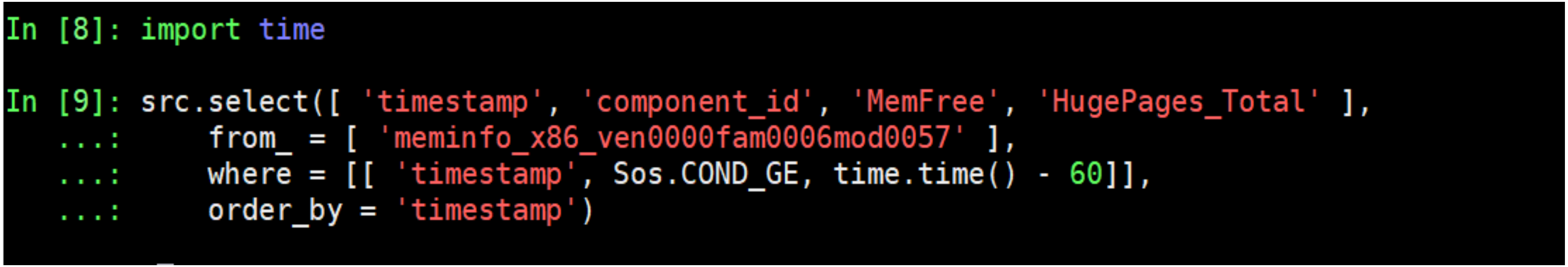
DataSource – show a particular schema



```
In [6]: src.show_schema('kokkos_app')
Name                             Id       Type         Indexed  Info
-------------------------------- -------- ------------ -------- --------------------------------
job_id                                  0 UINT64       True
job_name                                1 STRING       False
app_id                                  2 UINT64       False
inst_data                               3 STRUCT       False
start_time                              4 TIMESTAMP    False
mpi_rank                                5 UINT64       False
hostname                                6 STRING       False
user_id                                 7 UINT32       True
component_id                            8 UINT64       False
total_app_time                          9 DOUBLE       False
total_kernel_times                     10 DOUBLE       False
total_non_kernel_times                 11 DOUBLE       False
percent_in_kernels                     12 DOUBLE       False
unique_kernel_calls                    13 DOUBLE       False
inst_job_app_time                      14 JOIN         True     inst_data, job_id, app_id
```

DataSource – select data

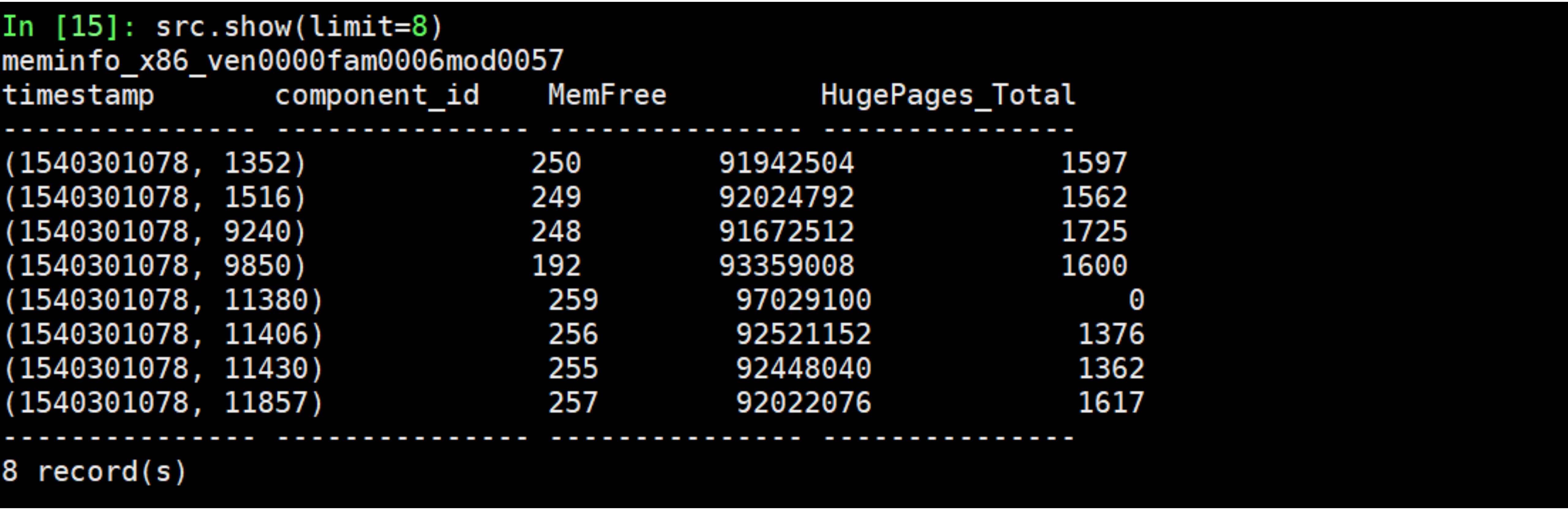
- *column* list specifies which data from schema is returned
- *from_* specifies which schema the data comes from
- *where* clause specifies select conditions
- *order_by* specifies the index



```
In [8]: import time

In [9]: src.select([ 'timestamp', 'component_id', 'MemFree', 'HugePages_Total' ],
   ...:     from_ = [ 'meminfo_x86_ven0000fam0006mod0057' ],
   ...:     where = [[ 'timestamp', Sos.COND_GE, time.time() - 60]],
   ...:     order_by = 'timestamp')
```

DataSource – show results

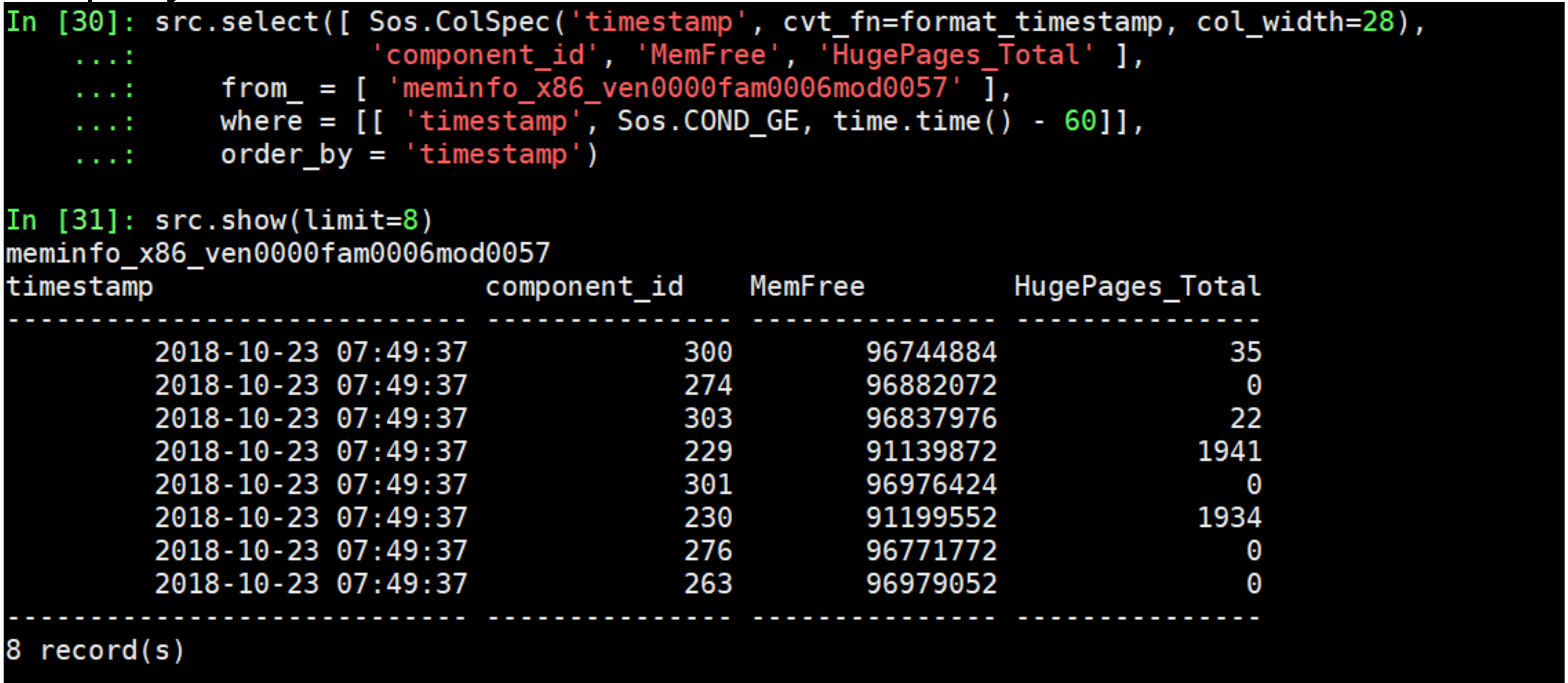
- Useful for verifying your select conditions during development
- Exploring the available data



```
In [15]: src.show(limit=8)
meminfo_x86_ven0000fam0006mod0057
timestamp       component_id    MemFree         HugePages_Total
--------------- --------------- --------------- ---------------
(1540301078, 1352)             250        91942504            1597
(1540301078, 1516)             249        92024792            1562
(1540301078, 9240)             248        91672512            1725
(1540301078, 9850)             192        93359008            1600
(1540301078, 11380)             259        97029100               0
(1540301078, 11406)             256        92521152            1376
(1540301078, 11430)             255        92448040            1362
(1540301078, 11857)             257        92022076            1617
--------------- --------------- --------------- ---------------
8 record(s)
```

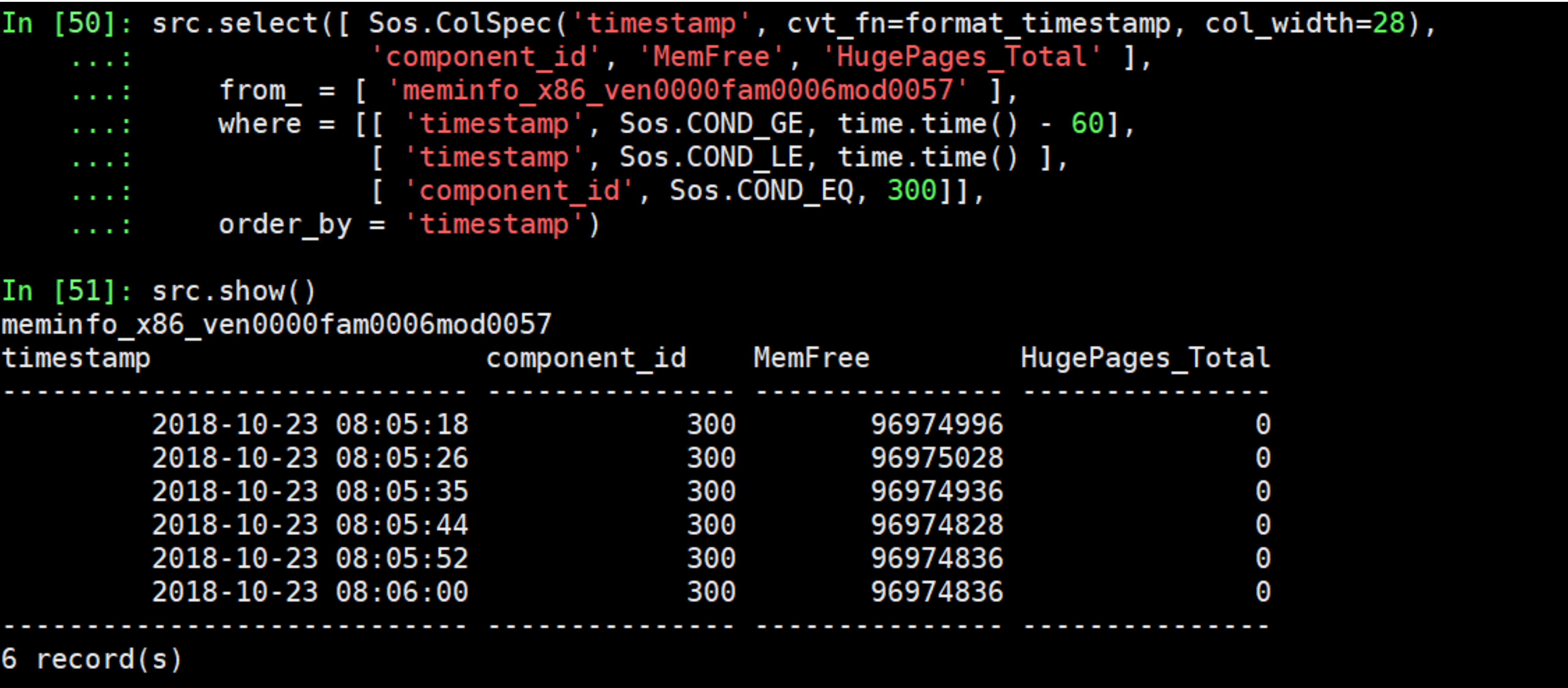
DataSource – controlling column formatting

- Columns can be formatted/transformed on input using a *column-specification*



```
In [30]: src.select([ Sos.ColSpec('timestamp', cvt_fn=format_timestamp, col_width=28),
    ...:              'component_id', 'MemFree', 'HugePages_Total' ],
    ...:     from_ = [ 'meminfo_x86_ven0000fam0006mod0057' ],
    ...:     where = [[ 'timestamp', Sos.COND_GE, time.time() - 60]],
    ...:     order_by = 'timestamp')

In [31]: src.show(limit=8)
meminfo_x86_ven0000fam0006mod0057
timestamp                    component_id    MemFree         HugePages_Total
---------------------------- --------------- --------------- ---------------
         2018-10-23 07:49:37             300        96744884              35
         2018-10-23 07:49:37             274        96882072               0
         2018-10-23 07:49:37             303        96837976              22
         2018-10-23 07:49:37             229        91139872            1941
         2018-10-23 07:49:37             301        96976424               0
         2018-10-23 07:49:37             230        91199552            1934
         2018-10-23 07:49:37             276        96771772               0
         2018-10-23 07:49:37             263        96979052               0
---------------------------- --------------- --------------- ---------------
8 record(s)
```

DataSource – *where* conditions are *ANDed*



```
In [50]: src.select([ Sos.ColSpec('timestamp', cvt_fn=format_timestamp, col_width=28),
    ...:              'component_id', 'MemFree', 'HugePages_Total' ],
    ...:     from_ = [ 'meminfo_x86_ven0000fam0006mod0057' ],
    ...:     where = [[ 'timestamp', Sos.COND_GE, time.time() - 60],
    ...:              [ 'timestamp', Sos.COND_LE, time.time() ],
    ...:              [ 'component_id', Sos.COND_EQ, 300]],
    ...:     order_by = 'timestamp')

In [51]: src.show()
meminfo_x86_ven0000fam0006mod0057
timestamp                    component_id    MemFree         HugePages_Total
---------------------------- --------------- --------------- ---------------
         2018-10-23 08:05:18             300        96974996               0
         2018-10-23 08:05:26             300        96975028               0
         2018-10-23 08:05:35             300        96974936               0
         2018-10-23 08:05:44             300        96974828               0
         2018-10-23 08:05:52             300        96974836               0
         2018-10-23 08:06:00             300        96974836               0
---------------------------- --------------- --------------- ---------------
6 record(s)
```

DataSet Class

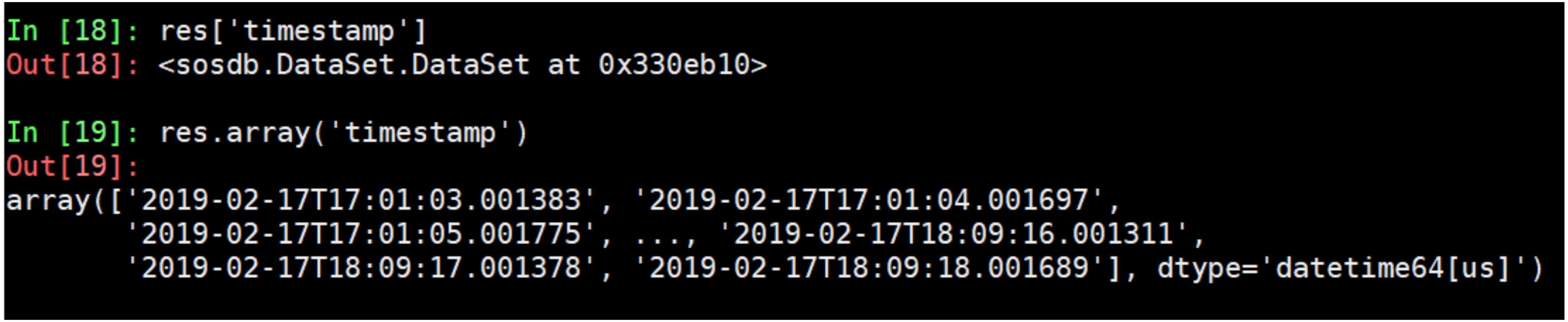
- Encapsulates data returned by a DataSource
- Intended to accelerate development of analysis by simplifying:
  - Accessing data series from a DataSource
  - Mathematical operations on data series
  - Combining data series together
- Design objectives:
  - Keep simple things simple
  - Make hard things easier

DataSet – a collection of data series

- Series in a DataSet are named*:*
  - matches it’s name in the DataSource.select statement, or
  - is specified directly by the programmer
- Internally, a series in a DataSet is a Numpy array
- All series in the same set have the same length
  - len(series) is the buffer size
  - series.get_series_size() is the amount of data stored in the buffer
- A series from a DataSet is accessed by *name* or by *index, e.g.*
  - timeSet = theSet[‘timestamp’]
  - timeSet = theSet[0]

DataSet – accessing series data

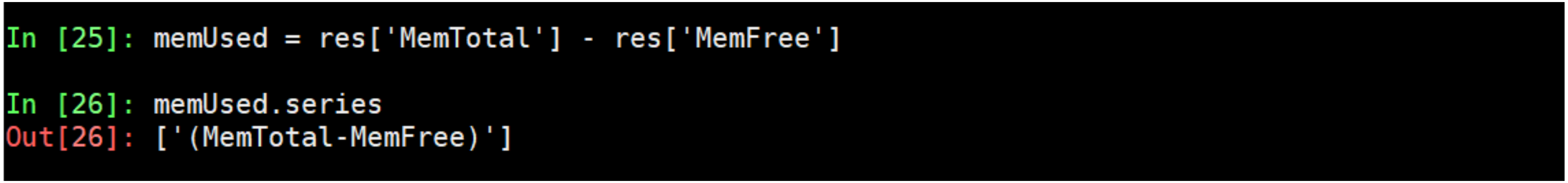
- dataset[”] returns another DataSet containing the series
- dataset.array(“) returns the numpy array for that series
- The 1st approach makes algebra easier, e.g.
    - cpi = res[‘PAPI_TOT_CYC’] / res[‘PAPI_TOT_INS’]



```
In [18]: res['timestamp']
Out[18]: <sosdb.DataSet.DataSet at 0x330eb10>

In [19]: res.array('timestamp')
Out[19]:
array(['2019-02-17T17:01:03.001383', '2019-02-17T17:01:04.001697',
       '2019-02-17T17:01:05.001775', ..., '2019-02-17T18:09:16.001311',
       '2019-02-17T18:09:17.001378', '2019-02-17T18:09:18.001689'], dtype='datetime64[us]')
```

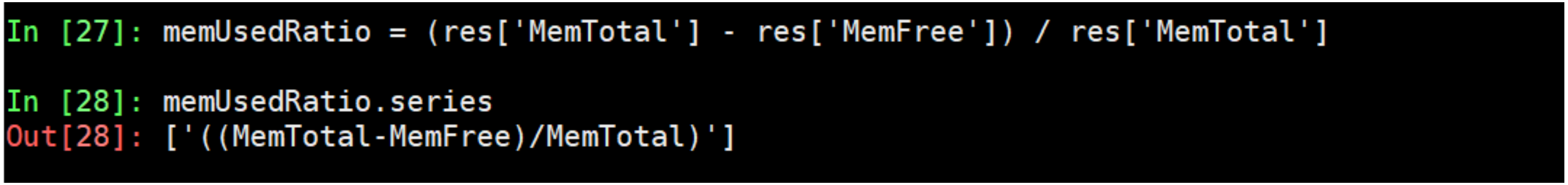
DataSet – algebraic result naming

- The name of a series that is the result of algebraic operations is “left-hand series name” “op” “right-hand series name”



```
In [25]: memUsed = res['MemTotal'] - res['MemFree']

In [26]: memUsed.series
Out[26]: ['(MemTotal-MemFree)']
```

- More complex expressions work as expected

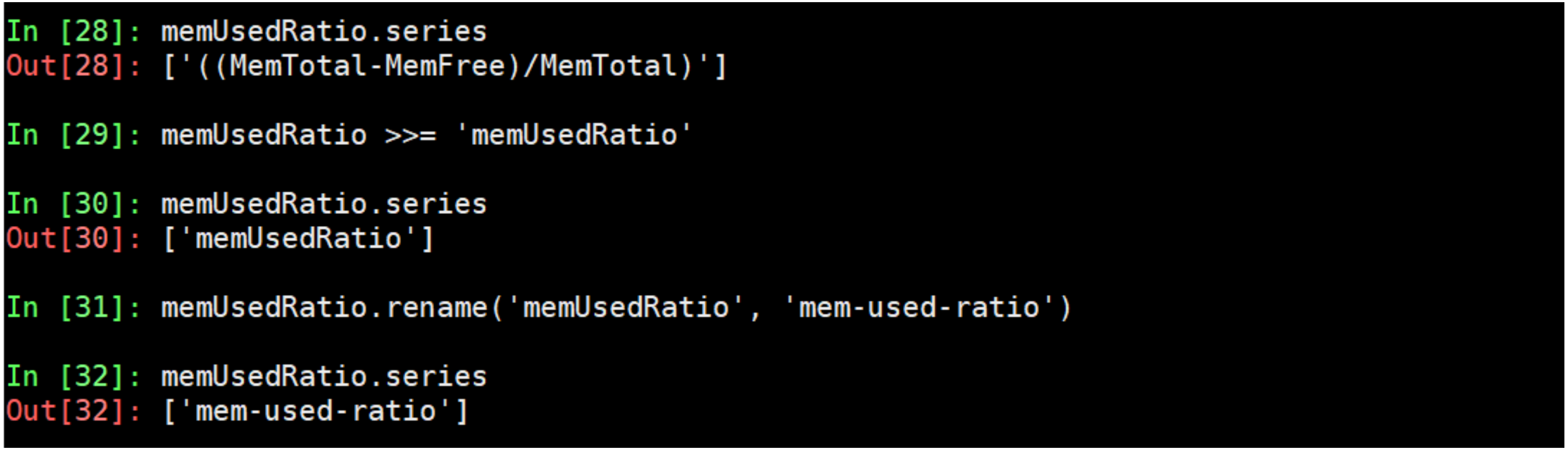


```
In [27]: memUsedRatio = (res['MemTotal'] - res['MemFree']) / res['MemTotal']

In [28]: memUsedRatio.series
Out[28]: ['((MemTotal-MemFree)/MemTotal)']
```

- Obviously, this could get messy ...

DataSet – Controlling the series names

- Series names can be renamed algebraically or functionally
  - res >>= ‘newName’
  - res.rename(‘oldName’, ‘newName’)

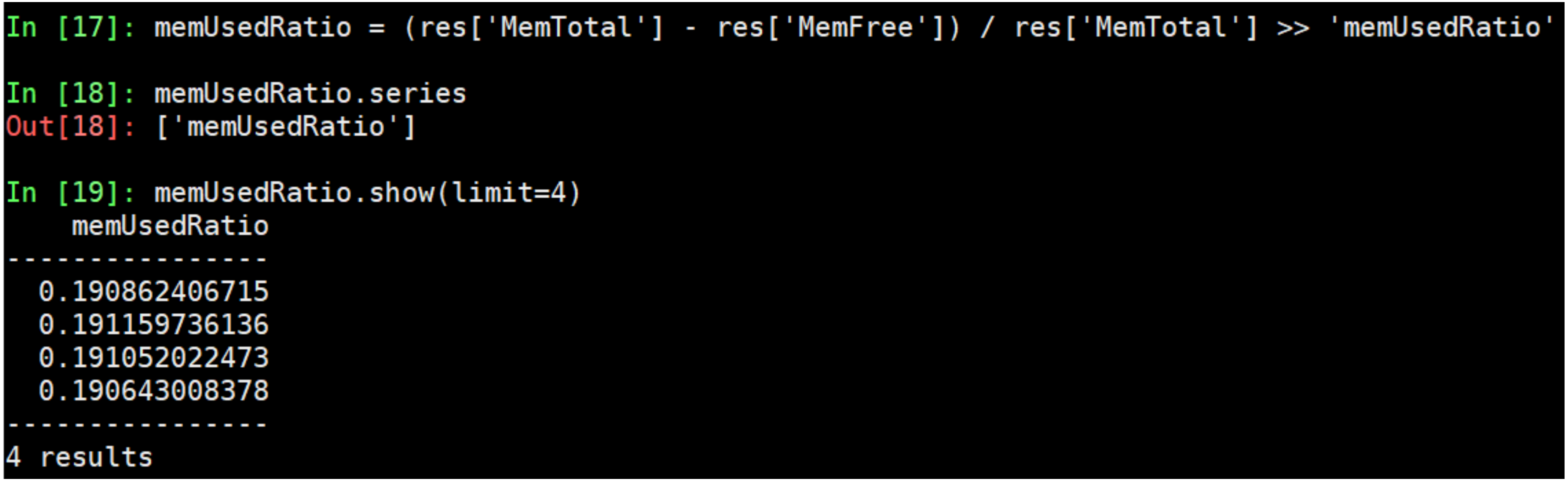


```
In [28]: memUsedRatio.series
Out[28]: ['((MemTotal-MemFree)/MemTotal)']

In [29]: memUsedRatio >>= 'memUsedRatio'

In [30]: memUsedRatio.series
Out[30]: ['memUsedRatio']

In [31]: memUsedRatio.rename('memUsedRatio', 'mem-used-ratio')

In [32]: memUsedRatio.series
Out[32]: ['mem-used-ratio']
```

- Use the >>= op when your DataSet contains only a single series
- Use the function when your DataSet contains many series

DataSet – Putting it all together

- All of these operations can be done in a single assignment expression

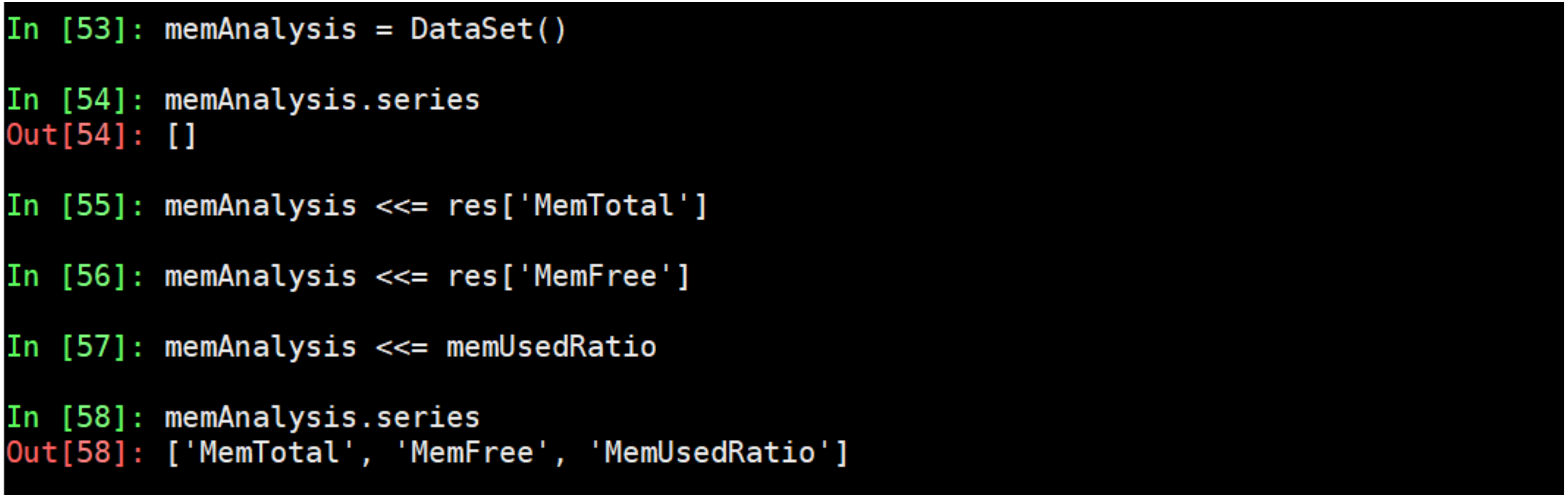


```
In [17]: memUsedRatio = (res['MemTotal'] - res['MemFree']) / res['MemTotal'] >> 'memUsedRatio'

In [18]: memUsedRatio.series
Out[18]: ['memUsedRatio']

In [19]: memUsedRatio.show(limit=4)
    memUsedRatio
----------------
  0.190862406715
  0.191159736136
  0.191052022473
  0.190643008378
----------------
4 results
```

- Typicaly this is how a result would be calculated

DataSet – Combining results

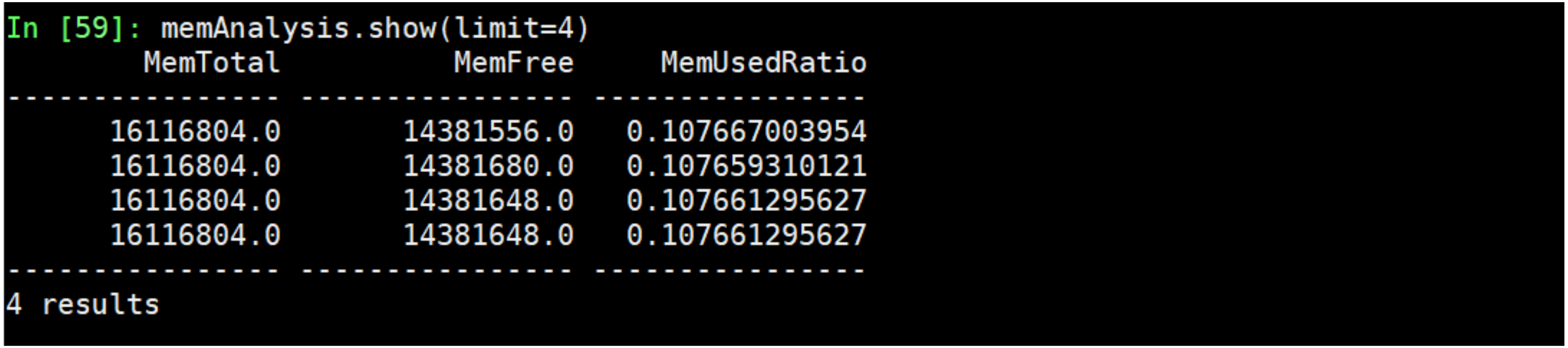
- DataSets can be combined together with the <<= operation



```
In [53]: memAnalysis = DataSet()

In [54]: memAnalysis.series
Out[54]: []

In [55]: memAnalysis <<= res['MemTotal']

In [56]: memAnalysis <<= res['MemFree']

In [57]: memAnalysis <<= memUsedRatio

In [58]: memAnalysis.series
Out[58]: ['MemTotal', 'MemFree', 'MemUsedRatio']
```

DataSet – Displaying Results

- DataSets can be displayed using the *show* method

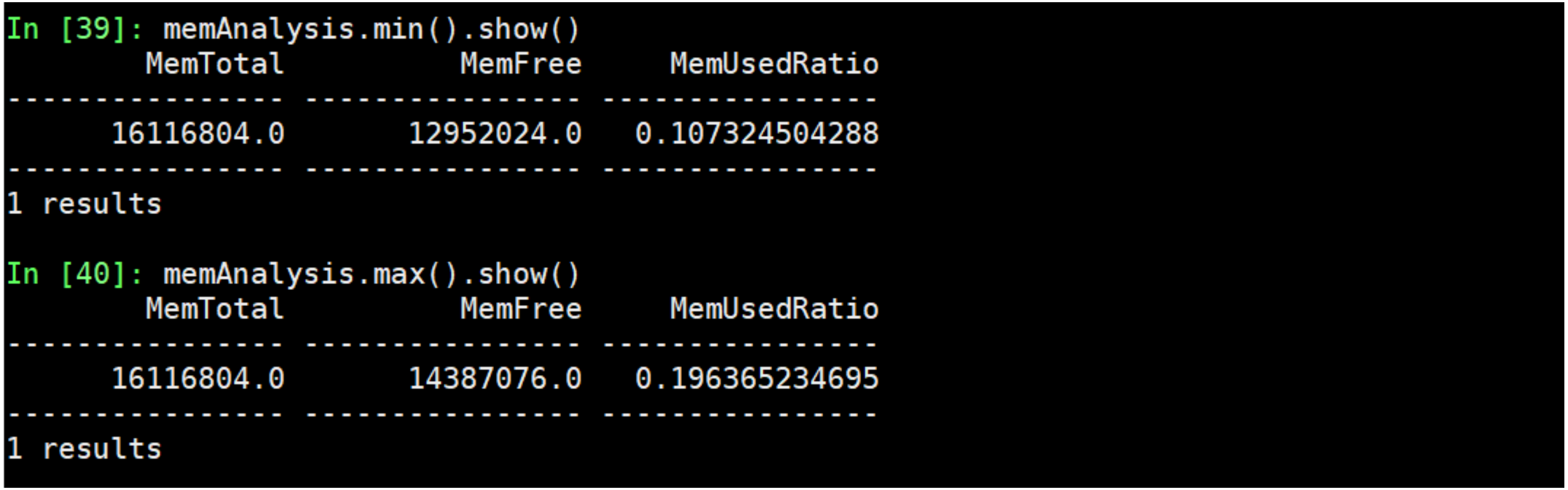


```
In [59]: memAnalysis.show(limit=4)
       MemTotal            MemFree      MemUsedRatio
---------------- ---------------- ----------------
     16116804.0        14381556.0   0.107667003954
     16116804.0        14381680.0   0.107659310121
     16116804.0        14381648.0   0.107661295627
     16116804.0        14381648.0   0.107661295627
---------------- ---------------- ----------------
4 results
```

- The *limit* parameter allows you to limit the number of rows shown

DataSet – min/max

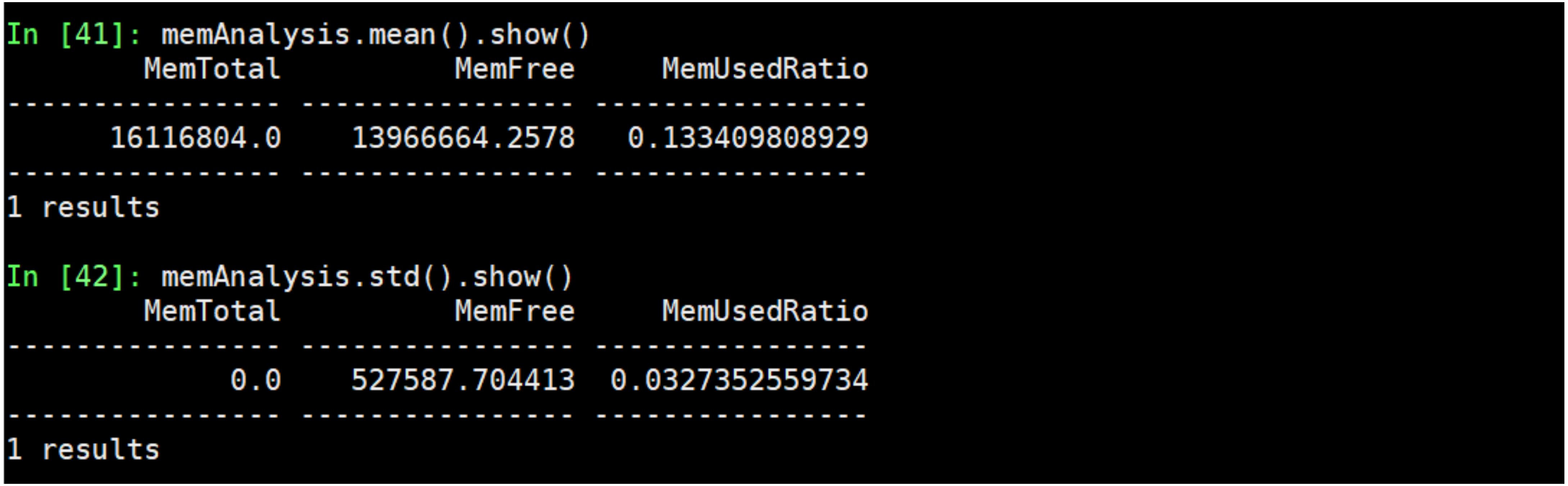
- DataSets support some common statistical operations



```
In [39]: memAnalysis.min().show()
        MemTotal                MemFree      MemUsedRatio
---------------- ---------------- ----------------
      16116804.0         12952024.0    0.107324504288
---------------- ---------------- ----------------
1 results

In [40]: memAnalysis.max().show()
        MemTotal                MemFree      MemUsedRatio
---------------- ---------------- ----------------
      16116804.0         14387076.0    0.196365234695
---------------- ---------------- ----------------
1 results
```

DataSet – mean/std

- DataSets support some common statistical operations



```
In [41]: memAnalysis.mean().show()
        MemTotal            MemFree     MemUsedRatio
---------------- ---------------- ----------------
      16116804.0    13966664.2578   0.133409808929
---------------- ---------------- ----------------
1 results

In [42]: memAnalysis.std().show()
        MemTotal            MemFree     MemUsedRatio
---------------- ---------------- ----------------
             0.0    527587.704413  0.0327352559734
---------------- ---------------- ----------------
1 results
```

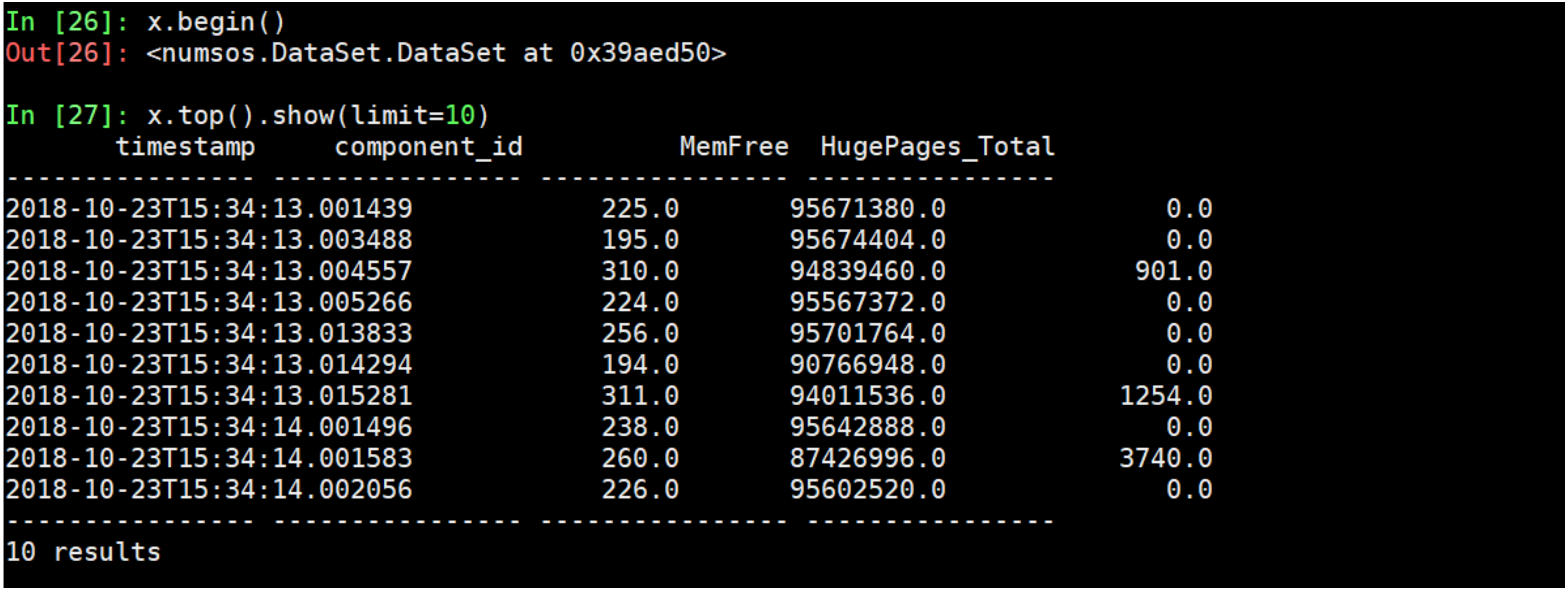
Transform Class

- A base class for DataSet vector operations
- Maintains a stack of DataSet
  - Transform operations pop arguments from the stack, and
  - Push results to the stack
- Implements a set of built-in transforms
  - Column-wise:
    - +, -, *, /
  - Row-wise:
    - histogram, 1st-difference, std, mean, min, max, etc…
    - Supports grouping of data by a series, for row-wise transforms
      - e.g. component_id, aries_rtr_id, etc…
  - Can be extended with new operations as required
- Simple, “intuitive” syntax:
  - x.dup()
  - x[‘-’]([‘MemTotal’, ‘MemFree’])
  - x.append(series=[‘MemTotal])
  - x[‘/’]([‘MemTotal-MemFree’ , ‘MemTotal’], result=‘Mem_Used_Ratio’)

Transform – *group-by* functionality

- Row-wise operations are challenging when series are interwoven in time, e.g.
  - $time_0, component_0, value_{00}$
  - $time_0, component_1, value_{10}$
  - $time_0, component_2, value_{20}$
  - $time_0, component_3, value_{30}$
  - …
- A 1st-difference of this is not *easy* because the values are not sequential in the array.
- The *group_name* parameter to the Transform row-wise functions performs this data management function ‘automatically’ regardless of the ordering of the group column in the input deck

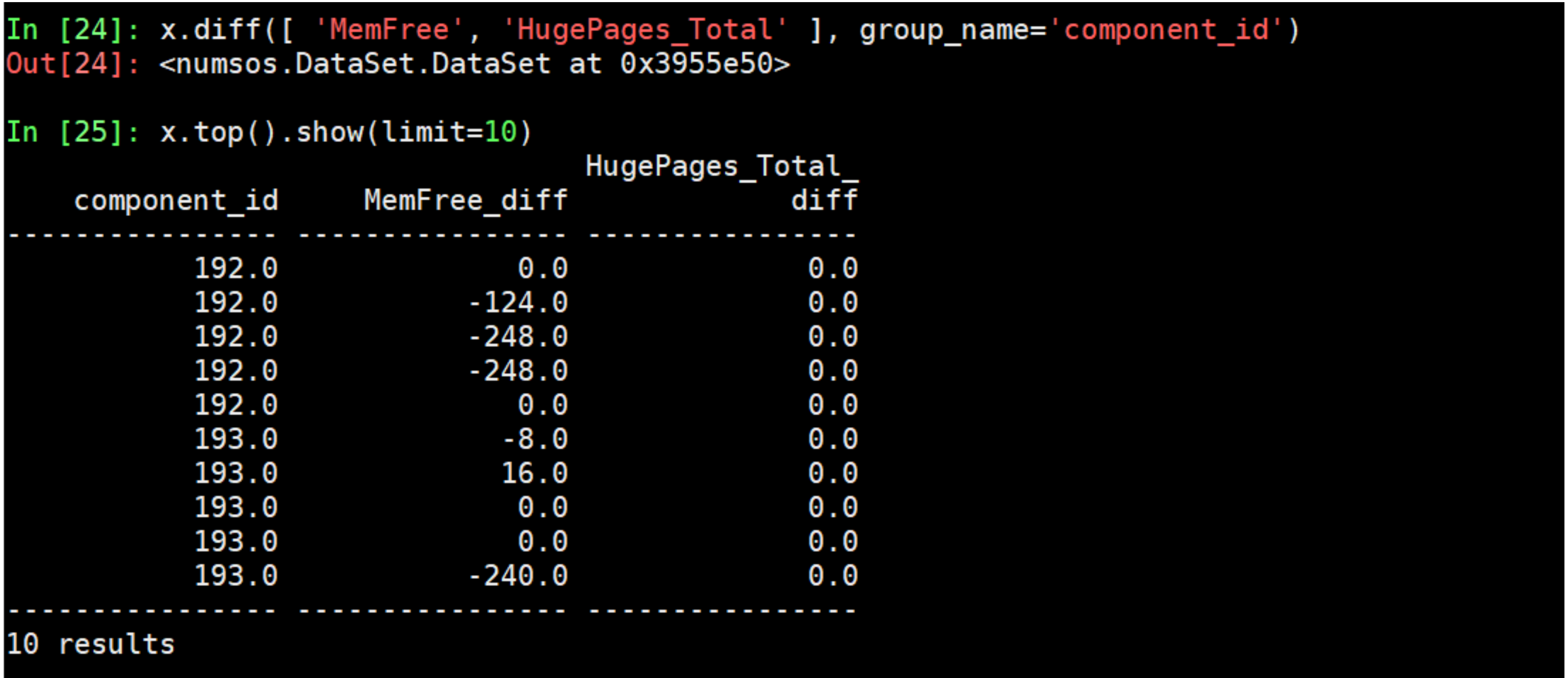
Transform – group-by functionality



```
In [26]: x.begin()
Out[26]: <numsos.DataSet.DataSet at 0x39aed50>

In [27]: x.top().show(limit=10)
       timestamp     component_id          MemFree  HugePages_Total
---------------- ---------------- ---------------- ----------------
2018-10-23T15:34:13.001439            225.0       95671380.0              0.0
2018-10-23T15:34:13.003488            195.0       95674404.0              0.0
2018-10-23T15:34:13.004557            310.0       94839460.0            901.0
2018-10-23T15:34:13.005266            224.0       95567372.0              0.0
2018-10-23T15:34:13.013833            256.0       95701764.0              0.0
2018-10-23T15:34:13.014294            194.0       90766948.0              0.0
2018-10-23T15:34:13.015281            311.0       94011536.0           1254.0
2018-10-23T15:34:14.001496            238.0       95642888.0              0.0
2018-10-23T15:34:14.001583            260.0       87426996.0           3740.0
2018-10-23T15:34:14.002056            226.0       95602520.0              0.0
---------------- ---------------- ---------------- ----------------
10 results
```

Transform – group-by functionality

- Post transform data organization
- Data ordered by group column



```
In [24]: x.diff([ 'MemFree', 'HugePages_Total' ], group_name='component_id')
Out[24]: <numsos.DataSet.DataSet at 0x3955e50>

In [25]: x.top().show(limit=10)
                                   HugePages_Total_
    component_id      MemFree_diff             diff
---------------- ---------------- ----------------
           192.0              0.0              0.0
           192.0           -124.0              0.0
           192.0           -248.0              0.0
           192.0           -248.0              0.0
           192.0              0.0              0.0
           193.0             -8.0              0.0
           193.0             16.0              0.0
           193.0              0.0              0.0
           193.0              0.0              0.0
           193.0           -240.0              0.0
---------------- ---------------- ----------------
10 results
```

Putting it all together

- PAPI Example ...

DataSink Class

- Ouput analog of the DataSource Class
- API designed to facilitate analysis that would work with CSV, SOS, or other data sources
- DataSink.config
  - Specify where and how data is to be stored
  - Parameters may be specific to each *back-end*
- DataSink.insert
  - SQL like syntax for identifying what data is to be stored
- DataSink.put_results
  - Store data from analysis

DataSink Class –CSV Example

```
# Configure the CSV data sink
csv = CsvDataSink()
csv.config(path="./netstat.csv", header=True)
csv.insert(
    [
        Sos.ColSpec("timestamp", cvt_fn=format_timestamp),
        Sos.ColSpec("component_id", cvt_fn=int),
        Sos.ColSpec("job_id", cvt_fn=int),
        "rx_bytes#p7p2_diff",
        "tx_bytes#p7p2_diff",
        "rx_packets#p7p2_diff",
        "tx_packets#p7p2_diff"
    ],
    into="netstat")
```

DataSink Class –SOS Example

```
# Configure the Sos data sink
sink = SosDataSink()
sink.config(path="/ORION_DATA/Mutrino_Results", create=True)
sink.insert(
    sink.Metric_Columns +
    [
        "rx_bytes#p7p2_diff",
        "tx_bytes#p7p2_diff",
        "rx_packets#p7p2_diff",
        "tx_packets#p7p2_diff"
    ],
    into = { "schema" : "netstat", "attrs" :
             sink.Metric_Attrs +
             [
                 { "name" : "rx_bytes#p7p2_diff",   "type" : "double" },
                 { "name" : "tx_bytes#p7p2_diff",   "type" : "double" },
                 { "name" : "rx_packets#p7p2_diff", "type" : "double" },
                 { "name" : "tx_packets#p7p2_diff", "type" : "double" }
             ]
             + sink.Metric_Joins
           }
    )
```

Streaming Analysis

- src = Configure the DataSource
- sink = Configure the DataSink
- x = Transform(src, sink)
- rc = x.begin()
- while rc:
    - x.this().and().that()
    - rc = x.next()