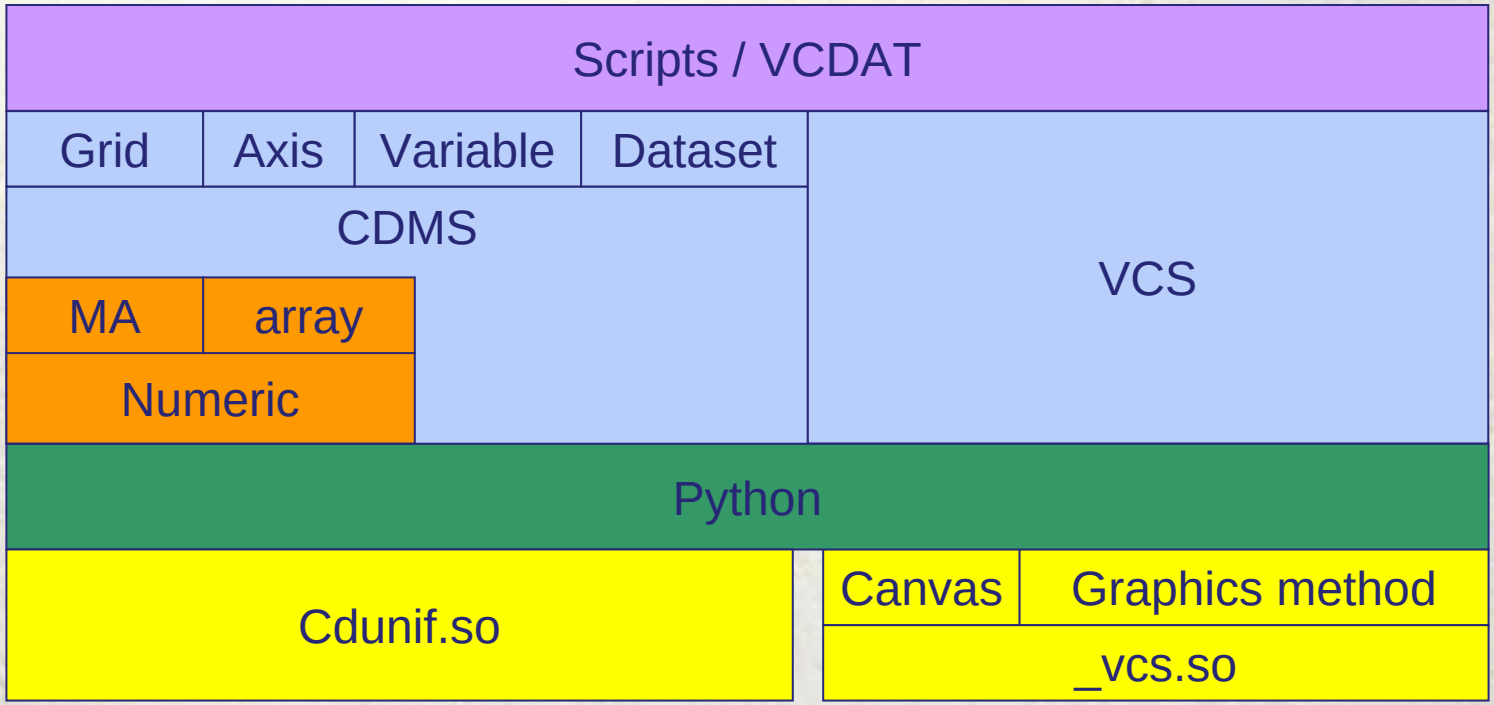
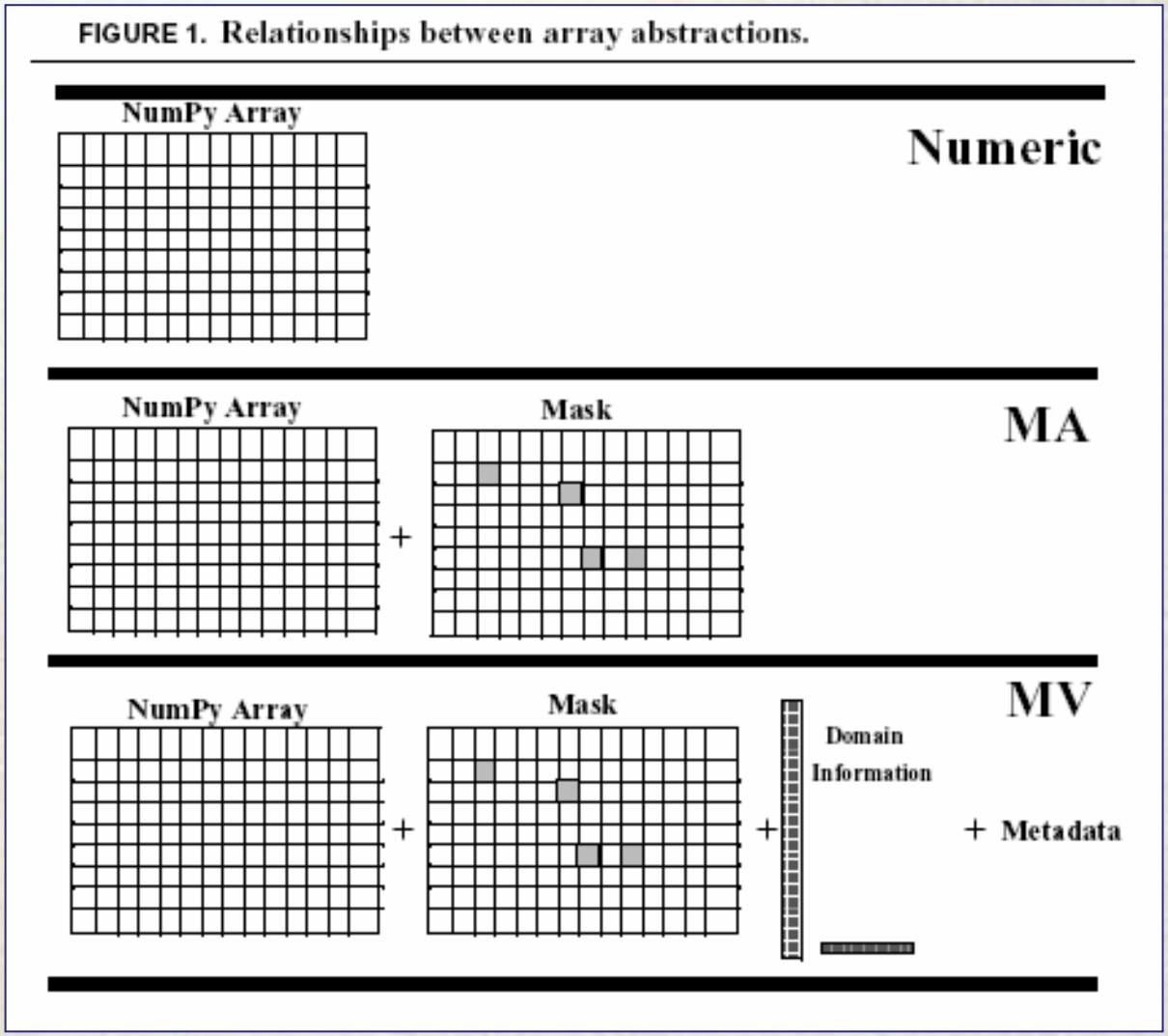
Advanced data manipulation

CDAT architecture



3 Types of Array



Numeric Arrays

```
>>> import Numeric

>>> a=Numeric.array([1,2,3,4,5])

>>> b=Numeric.array([[1.0,2.0,3.0,4.0],
... [1.5,2.5,3.5,4.5]])

>>> a
array([1, 2, 3, 4, 5])

>>> b
array([[ 1. ,  2. ,  3. ,  4. ],
       [ 1.5,  2.5,  3.5,  4.5]])
```

a				
1	2	3	4	5

b			
1.0	2.0	3.0	4.0
1.5	2.5	3.5	4.5

Basic arithmetic

```
>>> from Numeric import *

>>> a+10
array([11, 12, 13, 14, 15])

>>> b*array([[5,5,5,5],[2,2,2,2]])
array([[  5.,  10.,  15.,  20.],
       [  3.,   5.,   7.,   9.]])

>>> sin(a)
array([ 0.84147098,  0.90929743,
   0.14112001, -0.7568025 , -
   0.95892427])
```

a				
1	2	3	4	5

b			
1.0	2.0	3.0	4.0
1.5	2.5	3.5	4.5

Indexing and slicing

```
>>> b[0]
array([ 1.,  2.,  3.,  4.])

>>> b[0,2]
3.0
>>> b[0][2]
3.0

>>> b[1,-2]
3.5
>>> a[1:3]
array([2, 3])

>>> a[:5:2]
array([1, 3, 5])

>>> b[:2,:2]
array([[ 1. ,  2. ],
       [ 1.5,  2.5]])

>>> b[:2,::2]
array([[ 1. ,  3. ],
       [ 1.5,  3.5]])
```

a				
1	2	3	4	5

b			
1.0	2.0	3.0	4.0
1.5	2.5	3.5	4.5

Array Properties

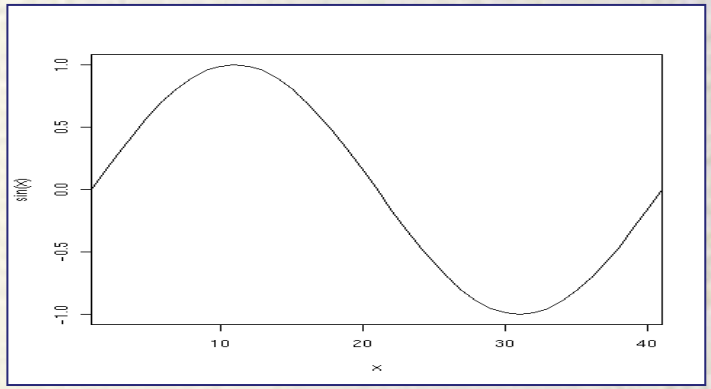
```
>>> a.shape
(5,)
>>> b.shape
(2, 4)

>>> b.typecode()
'd'
>>> b.typecode() == Float
True

>>> b.itemsize()
8

>>> a.byteswapped()
array([16777216, 33554432,
   50331648, 67108864, 83886080])
```

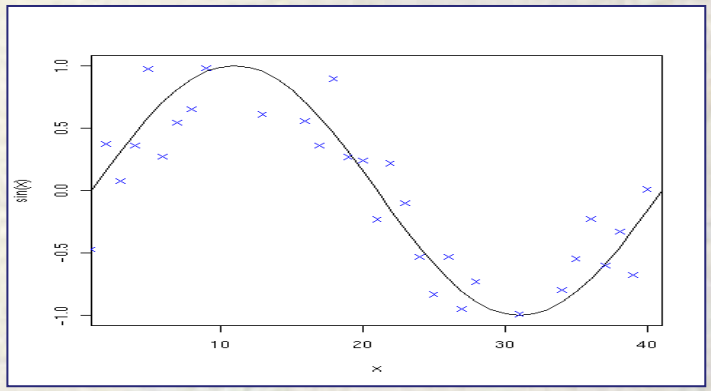
a				
1	2	3	4	5

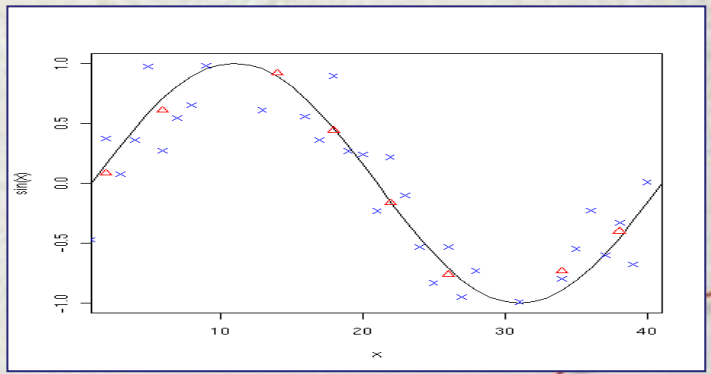
b			
1.0	2.0	3.0	4.0
1.5	2.5	3.5	4.5

A non trivial example

```
>>> from Numeric import *
>>> sin_x =
sin(arrayrange(0, 2*pi, pi/20))



>>> from RandomArray import *
>>> sin_x_r = (sin_x +
... random(sin_x.shape) - 0.5)



>>> bins = zeros(10)
>>> for x in range(4):
...     bins = bins + sin_x[x::4]
...
>>> bins = bins / 4
```





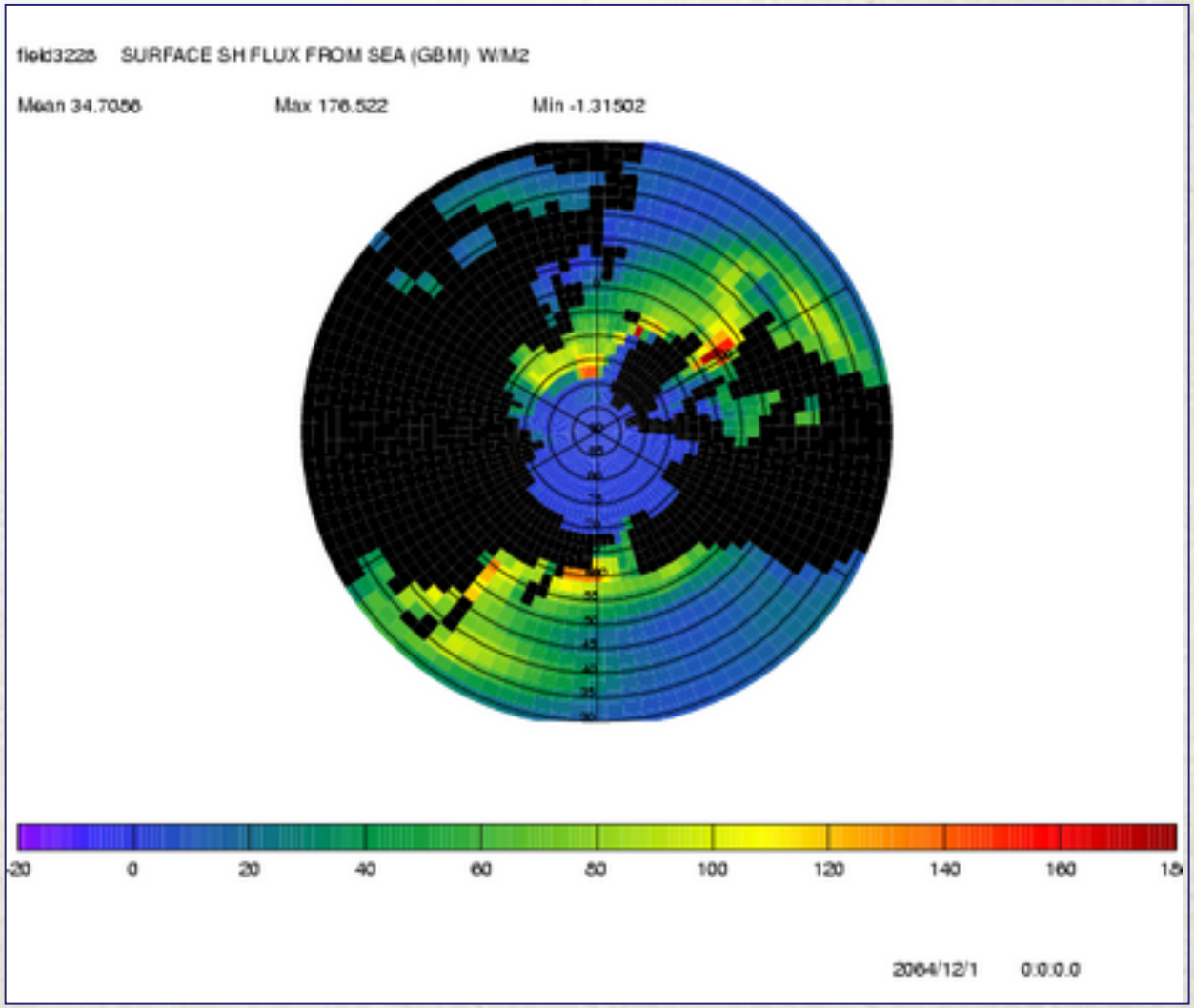


Numeric functions

- Too many to mention
  - http://numeric.scipy.org/numpydoc/numdoc.htm

take	put	putmask	transpose	fromstring
choose	ravel	nonzero	where	compress
diagonal	trace	product	outerproduct	argsort
argmax	argmin	repeat	array_repr	matrixmultiply
clip	indices	swapaxes	concatenate	innerproduct
sort	dot	array_str	resize	convolve
cumsum	identity	sum	cross_correlate	searchsorted
cumproduct	alltrue	sometrue	allclose	

Why we need masks



Creating masked arrays

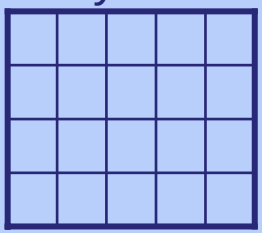
```
>>> import MA
>>> x = MA.array([1, 2, 3])


>>> y = MA.array([1, 2, 3],
... mask = [0, 1, 0])



>>> z = MA.masked_values([1.0,
   1.e20, 3.0, 4.0], 1.e20)



>>> z.mask()
[0,1,0,0,]
```

- Use MA as a replacement for Numeric:
- To create an array with the second element invalid, we would do
- To create a masked array where all values "near" 1.e20 are invalid, we can do
- The mask is stored as a separate array.

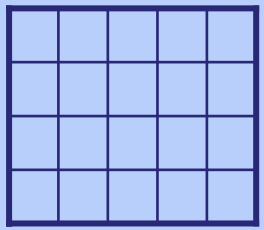
Anatomy of a CDAT Masked Variable

Masked Variable

array



mask



axis1

axis2

array metadata

key	value

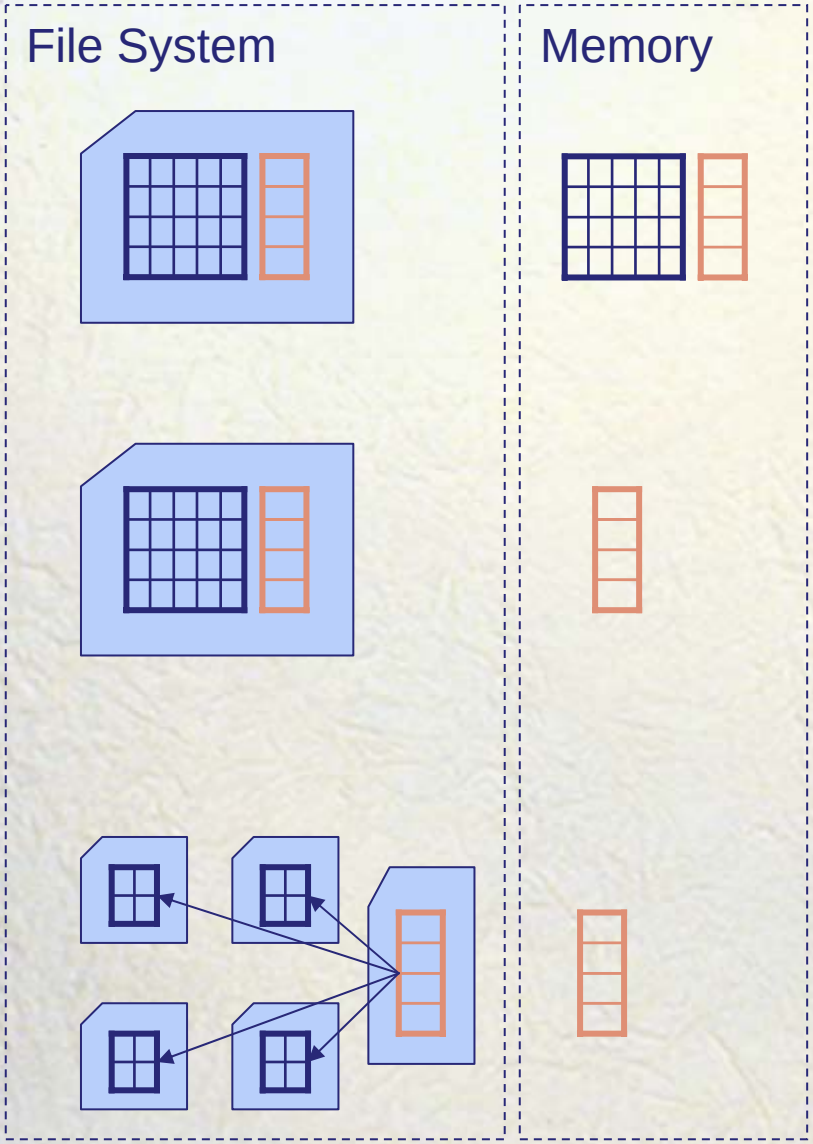
axis1 metadata

key	value

axis2 metadata

key	value

3 Types of variables



- Transient variable
  - All data copied to memory
- File variable
  - Metadata copied to memory
  - Data accessed in situ
  - Read or Read/Write access
- Dataset variable
  - Data distributed between multiple files
  - Read access only

Transient variables

```
>>> f = cdms.open(‘afile’, ‘a’)
>>> var = f(‘tas’)

>>> var.listattributes()
['comment', 'units',
   'level_description', 'subgrid',
   'long_name', 'grid_name', ...
>>> var.long_name
'Surface (1.5m) air temperature'

>>> var.shape
(4, 73, 96)
>>> var[3,0:2,0:2]
tas
array(
 [[ 245.2923584 , 245.2923584 ,]
 [ 246.42282104, 246.51434326,]])
```

- Use ( ) to create a transient variable from a file.
- List metadata
- Behaves like an array

File Variables

```
>>> f = cdms.open('afile', 'a')
>>> var = f['tas']

>>> var.long_name
'Surface (1.5m) air temperature'

>>> var.shape
(4, 73, 96)
>>> var[3,0:2,0:2]
tas
array(
 [[ 245.2923584 , 245.2923584 ,]
 [ 246.42282104, 246.51434326,]])

>>> var[3,0:2,0:2] =
   array([[1.,2.],[3.,4.]])
>>> f.close()
```

- Use [ ] to create a file variable.
- Standard MV.array features are accessible.
- Assigning to a slice will change data on disk

MV Example with masks

```
>>> import cdms, MV
>>> f_surface = cdms.open('sftlf_ta.nc')
>>> surf = f_surface('sftlf')

# Designate land where "surf" has values
# not equal to 100
>>> land_only = MV.masked_not_equal(surf, 100.)
>>> land_mask = MV.getmask(land_only)

# Now extract a variable from another file
>>> f = cdms.open('ta_1994-1998.nc')
>>> ta = f('ta')

# Apply this mask to retain only land values.
>>> ta_land = cdms.createVariable(ta, mask=land_mask,
... copy=0, id='ta_land')
```

Axes

```
>>> lat = f['latitude']
        OR
>>> lat = var.getLatitude()
>>> lat
   id: latitude
   Designated a latitude
   axis.
   units:  degrees_north
   Length: 73
   First:  90.0
   Last:   -90.0
   Other axis attributes:
      long_name: latitude
      axis: Y
   Python id:  b707d38c
```

- Like CF NetCDF, axes are stored as variables.
- Variables know their axes.
- Axes have some but not all MV.array features

Creating a good axis from scratch

```
>>> values=range(0,360,5)
>>> lon=cdms.createAxis(values)



>>> lon.designateLongitude()




>>> lon.id="longitude"
>>> lon.standard_name="longitude"
>>> lon.units="degrees_east"
>>> lon.comment="This really is
   longitude!"
```

- Create an array from a list or Numeric
- You could stop here, but we like metadata! So designate it
- And name, units…

Creating a CDMS variable

- You need to use **cdms.createVariable():**

```
cdms.createVariable(array,
  typecode=None, copy=0, savespace=0,
  mask=None, fill_value=None,
  grid=None, axes=None,
  attributes=None, id=None)
```

- See the CDMS manual for a full explanation of the options:

  http://www-pcmdi.llnl.gov/software-portal/cdat/documentation/manuals/cdms.pdf

Many ways to subset

- With MV we have two ways of referencing subsets
  - "index space", [start:stop:stride]. Just like standard arrays.
  - "coordinate space". Using axis names and values.

```
var[start:stop:stride]


var(time=slice(start, stop, stride))


var(time=(min, max))


var(latitude=(min,max),
    longitude=(min, max))

file(varname, time=(min,max))
```

- Standard "index space" subsetting
- "index space" subsetting with axis selection.
- "coordinate space" with axis range.
- select on multiple axes
- Direct subsetting from dataset object.

Selectors – another way of sub-setting

- Define a selector that can then be re-used in code:

```
from cdms.selectors import Selector
sel1 = Selector(time=('1979-1-1','1979-2-1'),
  level=1000.)
x1 = v1(sel1)
x2 = v2(sel1)
```

- Pre-defined selector slices for axes:

```
from cdms import timeslice, levelslice
x = hus(timeslice(0,2), levelslice(16,17))
```

- Or you can use the **domain** selectors in **cdutil**:

```
from cdutil.region import *
NH=NorthHemisphere=domain(latitude=(0.,90.))
SH=SouthHemisphere=domain(latitude=(-90.,0.))
```

A Tip for diagnosing problems

- Sometimes it's not obvious which type of object you are using

```
>>> a = pr.getValue()
>>> a
array(
 array (12,73,96) , type = f, has
   84096 elements)
>>> type(a)
<class 'MA.MA.MaskedArray'>


>>> type(pr)
<type 'instance'>
>>> pr
<Variable: pr, dataset: none,
   shape: (12, 73, 96)>
```

- Use type(obj) to discover what you have.
- This isn't very helpful for fileVariable and Dataset objects
- Datasets have a "dataset" property, files a "file property.