Notes here: http://goo.gl/P55tZ ¶

In[1]:

```
# First, we import requied packages
import ftplib # ftplib is an FTP client library
import numpy as np # numpy is quite useful

nasa_ftp =  "e4ftl01.cr.usgs.gov" # The location of the NASA FTP site
ftp = ftplib.FTP( nasa_ftp )
ftp.login("anonymous", "thisisme") # Username and password. Password is usually your email address
ftp.cwd( 'MODIS_Composites/MOTA/MCD15A2.005' ) # Change directory
data = []
ftp.dir(data.append) # Get the listing. Note that this requires a method to fill each line
```

In[2]:

```
print data # Output is
```

```
['total 59664', 'drwxr-xr-x  2 90 122880 Jun 20  2008 2002.07.04', 'drwxr-xr-x  2 90 116736 Jun 20  2008 2002.07
```

Parsing the directory structue

We want extract the information in the directory names so we can select the ones we are interested in

We want to store the year, month and day as integers

Since each directory line is a string, we can use string methods to **split** the bits we want

But we need a robust way of doing this, as there can be **rogue directory names** popping up

In[4]:

```
all_dirs = [] # We shall store the directory dates here
for this_line in data[1:]: # We can loop over the directory listing lines
                           # Note we are skipping the first element
    the_dir = this_line.split()[-1] # Split by whitespace and grab the last element, the directory name
    parsed_dir = [] # A temporary empty list
    if ( len(the_dir.split(".")) == 3 ): # If we have three elements after splitting by "."
        for k in the_dir.split("."): # Loop over the three elements
            if k.isdigit(): # Test whether we have a digit
                parsed_dir.append ( int(k) ) # We, do, convert the string to digit and append to temporary list
            else: # We don't
                parsed_dir.append ( 0 ) # Append a 0
    else:
        parsed_dir = [0,0,0]
    all_dirs.append ( parsed_dir ) # Append temporary list to final list
```

Catching errors: Exceptions

The above code is quite messy (the one in the notes is even worse!), and not very robust. What if the line

```
the_dir = this_line.split()[-1]
```

fails? In practice, there could be all sorts of other errors. We can use **exceptions** to catch these. We have the following structure

```
try:
    # Do something that might break
except:
    # Alternative action in case something broke in the try block
```

We can even specify different *exception types* to react to different errors.

In the previous case, we can make the code more succint using exceptions.

REMEMBER cleaner and leaner code results in simpler debugging!

In[5]:

```
all_dirs = []
for this_line in data:
    try:
        the_dir = this_line.split()[-1].split(".")
        parsed_dir = [ int(the_dir[0]), int(the_dir[1]), int(the_dir[2])]
    except:
        parsed_dir = [ 0, 0, 0 ]
    all_dirs.append ( parsed_dir )

print all_dirs
```

```
[[0, 0, 0], [2002, 7, 4], [2002, 7, 12], [2002, 7, 20], [2002, 7, 28], [2002, 8, 5], [2002, 8, 13], [2002, 8, 21
```

Selecting directories

Let's say you want to access data only for a given year and month... How do we go about that?

The simplest way is by looping:

In[6]:

```
the_year = 2011
the_month = 7
for the_dir in all_dirs:
    if ( the_dir[0] == the_year ) and ( the_dir[1] == the_month ):
```

```
        print the_dir
        # Or actually do something interesting
```

```
[2011, 7, 4]
[2011, 7, 12]
[2011, 7, 20]
[2011, 7, 28]
```

We can also use the fact that `all_dirs` is a list, convert it into a numpy array and use numpy to index it

In[7]:

```
all_dirs = np.array ( all_dirs ) # You make an array out of a list by using np.array( <list name> )
print all_dirs.shape
selected_dirs1 = np.logical_and ( all_dirs[:, 0] == the_year, all_dirs[:,1] == the_month )
print all_dirs[selected_dirs1] # Question: what is selected_dirs1?
# We can also use the np.where construct
selected_dirs2 = np.where ( (all_dirs[:, 0] == the_year) * ( all_dirs[:,1] == the_month ) )[0]
print all_dirs[selected_dirs2] # Question: what is selected_dirs?
```

```
(475, 3)
[[2011    7    4]
 [2011    7   12]
 [2011    7   20]
 [2011    7   28]]
[[2011    7    4]
 [2011    7   12]
 [2011    7   20]
 [2011    7   28]]
```

In[8]:

```
print selected_dirs1
print selected_dirs2
print np.array( data )[selected_dirs2]
```

```
[False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False  True  True  True  True False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False]
[415 416 417 418]
['drwxr-xr-x  2 90 237568 Aug 22  2011 2011.07.04'
 'drwxr-xr-x  2 90 114688 Jul 25  2011 2011.07.12'
 'drwxr-xr-x  2 90 112640 Jul 29  2011 2011.07.20'
```

```
 'drwxr-xr-x  2 90 122880 Aug  9  2011 2011.07.28']
```

Downloading stuff

So, what do we need to do to download a file?

1. We need to change into the required directory
2. We need to download the file
3. Go back to #1 and repeat

In order to download a file from FTPLIB, we need to have a **handler function**, that sends the data to an already opened file.

Also, we shall only download files from a single MODIS "tile", h09v05. See where this tile lies

![MODIS tile map] (http://nsidc.org/data/docs/daac/mod10_modis_snow/images/sinusoidal.gif)

In[9]:

```
def write_chunk ( block ):
    """A file writer handler for FTPLIB. Writes `block`
    to an open file handler called `fp`"""
    fp.write ( block )
# Need to select the actual directories...
the_dirs = np.array( data )[selected_dirs2]
# We only want a small portion of the surface of the Earth
tile = "h09v05"

quick_looks = [] # There are also a number of JPG quicklooks. Let's track their filenames

for curr_line in the_dirs: # Loop over the desired directories/folders in the remote FTP server
    curr_dir = curr_line.split()[-1] # Just the dirname
    ftp.cwd ( curr_dir ) # Change directory to curr_dir
    file_list = [] # Each folder has a file list
    ftp.dir( file_list.append ) # Now stored in file_list
    for fich in file_list: # Loop over the elements of the file list
        if fich.find ( tile ) >= 0: # If we find the name of the tile in the filename...
```

```
            the_fich = fich.split()[7] # Grab the actual filename

            if the_fich.find ( ".1.jpg" ) >= 0: # If the filename has a ".1.jpg" extension, it's a
                quick_looks.append ( the_fich ) # quicklook file

            print "Downloading %s" % the_fich # let the user know what's happening

            fp = open ( the_fich, 'w' ) # Open the output file
            ftp.retrbinary ( "RETR %s" % the_fich, write_chunk ) # Do the downloading. Use the handler
                                                                 # defined above
            fp.close() # Close the file for writing
            print "\t>>> Done!"

    ftp.cwd ( ".." ) # Go back to parent directory in remote server
```

```
Downloading BROWSE.MCD15A2.A2011185.h09v05.005.2011213154534.1.jpg
    >>> Done!
Downloading BROWSE.MCD15A2.A2011185.h09v05.005.2011213154534.2.jpg
    >>> Done!
Downloading MCD15A2.A2011185.h09v05.005.2011213154534.hdf
    >>> Done!
Downloading MCD15A2.A2011185.h09v05.005.2011213154534.hdf.xml
    >>> Done!
Downloading BROWSE.MCD15A2.A2011193.h09v05.005.2011203175347.1.jpg
    >>> Done!
Downloading BROWSE.MCD15A2.A2011193.h09v05.005.2011203175347.2.jpg
    >>> Done!
Downloading MCD15A2.A2011193.h09v05.005.2011203175347.hdf
    >>> Done!
Downloading MCD15A2.A2011193.h09v05.005.2011203175347.hdf.xml
    >>> Done!
Downloading BROWSE.MCD15A2.A2011201.h09v05.005.2011210161955.1.jpg
    >>> Done!
Downloading BROWSE.MCD15A2.A2011201.h09v05.005.2011210161955.2.jpg
    >>> Done!
Downloading MCD15A2.A2011201.h09v05.005.2011210161955.hdf
    >>> Done!
Downloading MCD15A2.A2011201.h09v05.005.2011210161955.hdf.xml
    >>> Done!
Downloading BROWSE.MCD15A2.A2011209.h09v05.005.2011220122803.1.jpg
    >>> Done!
Downloading BROWSE.MCD15A2.A2011209.h09v05.005.2011220122803.2.jpg
    >>> Done!
```

```
Downloading MCD15A2.A2011209.h09v05.005.2011220122803.hdf
    >>> Done!
Downloading MCD15A2.A2011209.h09v05.005.2011220122803.hdf.xml
    >>> Done!
```

Seeing the result of our exploit

You'll notice we now have some JPG images. We can use the shell program `display <filename>` to view them interactively. We can also plot them with matplotlib..

In[12]:

```
from IPython.core.display import Image
Image(filename='BROWSE.MCD15A2.A2011185.h09v05.005.2011213154534.1.jpg')
```

Out[12]:

```
<IPython.core.display.Image at 0xaacf8cc>
```

Examining the MODIS HDF files in the shell

The GDAL tools offer many command line utilitis to perform many operations with raster and vector spatial datasets, see here.

Let's see what we can find out about the files from the shell, and we'll then use the Python GDAL bindings to read data into Python and manipulate it.

Open a shell and `cd` to the directory/folder where you can see the HDF files you've just downloaded...

Then just use the gdalinfo program to examine the HDF dataset

```
$ gdalinfo MCD15A2.A2011185.h09v05.005.2011213154534.hdf
Driver: HDF4/Hierarchical Data Format Release 4
Files: MCD15A2.A2011185.h09v05.005.2011213154534.hdf
Size is 512, 512
Coordinate System is ``
Metadata:
[...]
VERTICALTILENUMBER=05
WESTBOUNDINGCOORDINATE=-117.486656023174
Subdatasets:
  SUBDATASET_1_NAME=HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2:Fpar_1km
  SUBDATASET_1_DESC=[1200x1200] Fpar_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)
  SUBDATASET_2_NAME=HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2:Lai_1km
  SUBDATASET_2_DESC=[1200x1200] Lai_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)
  SUBDATASET_3_NAME=HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2:FparLai_Q
  SUBDATASET_3_DESC=[1200x1200] FparLai_QC MOD_Grid_MOD15A2 (8-bit unsigned integer)
  SUBDATASET_4_NAME=HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2:FparExtra
  SUBDATASET_4_DESC=[1200x1200] FparExtra_QC MOD_Grid_MOD15A2 (8-bit unsigned integer)
  SUBDATASET_5_NAME=HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2:FparStdDe
  SUBDATASET_5_DESC=[1200x1200] FparStdDev_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)
  SUBDATASET_6_NAME=HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2:LaiStdDev
  SUBDATASET_6_DESC=[1200x1200] LaiStdDev_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0,  512.0)
Upper Right (  512.0,    0.0)
Lower Right (  512.0,  512.0)
Center      (  256.0,  256.0)
```

We have **6 datasets**:

- fAPAR @ 1km
- LAI @ 1 km
- fAPAR \(σ\)
- LAI \(σ\)
- Standard QA (quality assurance) flags
- Extended QA flags

We can get more insight into each dataset by using its GDAL name (see after the = symbol where it says `SUBDATASET_1_NAME`, `SUBDATASET_2_NAME`, etc). We need to use single quotes `'` around the GDAL dataset:

```
gdalinfo  'HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2:Fpar_1km'
Driver: HDF4Image/HDF4 Dataset
Files: MCD15A2.A2011185.h09v05.005.2011213154534.hdf
Size is 1200, 1200
Coordinate System is:
PROJCS["unnamed",
    GEOGCS["Unknown datum based upon the custom spheroid",
        DATUM["Not specified (based on custom spheroid)",
            SPHEROID["Custom spheroid",6371007.181,0]],
        PRIMEM["Greenwich",0],
        UNIT["degree",0.0174532925199433]],
    PROJECTION["Sinusoidal"],
    PARAMETER["longitude_of_center",0],
    PARAMETER["false_easting",0],
    PARAMETER["false_northing",0],
    UNIT["Meter",1]]
Origin = (-10007554.676999999210238,4447802.078666999936104)
Pixel Size = (926.625433055833014,-926.625433055833355)
Metadata:
[...]
Corner Coordinates:
Upper Left  (-10007554.677, 4447802.079) (117d29'11.96"W, 40d 0' 0.00"N)
Lower Left  (-10007554.677, 3335851.559) (103d55'22.97"W, 30d 0' 0.00"N)
Upper Right (-8895604.157, 4447802.079) (104d25'57.30"W, 40d 0' 0.00"N)
Lower Right (-8895604.157, 3335851.559) ( 92d22'33.76"W, 30d 0' 0.00"N)
Center      (-9451579.417, 3891826.819) (103d45'57.02"W, 35d 0' 0.00"N)
Band 1 Block=1200x100 Type=Byte, ColorInterp=Gray
      Description = MCD15A2 MODIS/Terra+Aqua Gridded 1KM FPAR (8-day composite)
      NoData Value=255
      Unit Type: Percent
      Offset: 0,   Scale:0.01
```

Note how this tells us

- the size of the dataset
- the projection used (in this case, the MODIS sinusoidal projection)
- the pixel size

- the origin of the dataset (upper left corner)
- lots of “metadata”
- the coordinates of the corners of the raster
- the bands, as well as their datatype

Opening the dataset in python

Just to list the available datasets in the HDF file, we can open the filename, and use the `GetSubDatasets()` method...

In[28]:

```
from osgeo import gdal
g = gdal.Open ( "MCD15A2.A2011185.h09v05.005.2011213154534.hdf")
sds = g.GetSubDatasets()
print sds
```

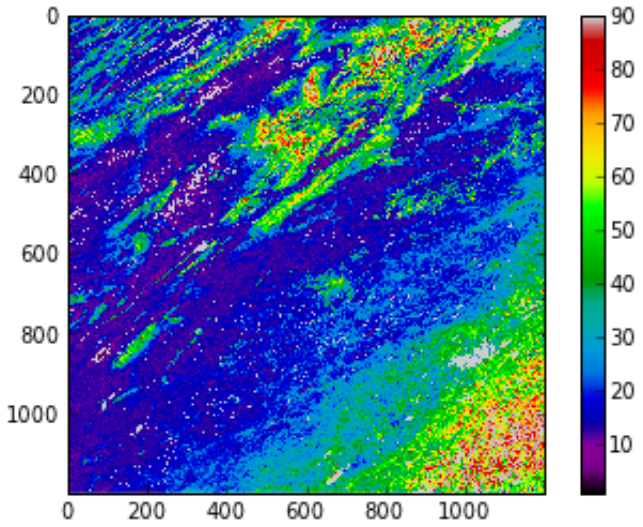
```
[('HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2:Fpar_1km', '[1200x1200] Fp
```

If we use the actual GDAL name, we will actually open the raster data....

In[20]:

```
g = gdal.Open ( 'HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2:Fpar_1km' )
fapar = g.ReadAsArray()
plt.imshow ( fapar, vmin=1, vmax=90, interpolation='nearest', cmap=plt.cm.spectral )
plt.colorbar()
print "The type of fapar is " , type ( fapar )
print "The size of fapar is ", fapar.shape
```

```
The type of fapar is  <type 'numpy.ndarray'>
The size of fapar is  (1200, 1200)
```

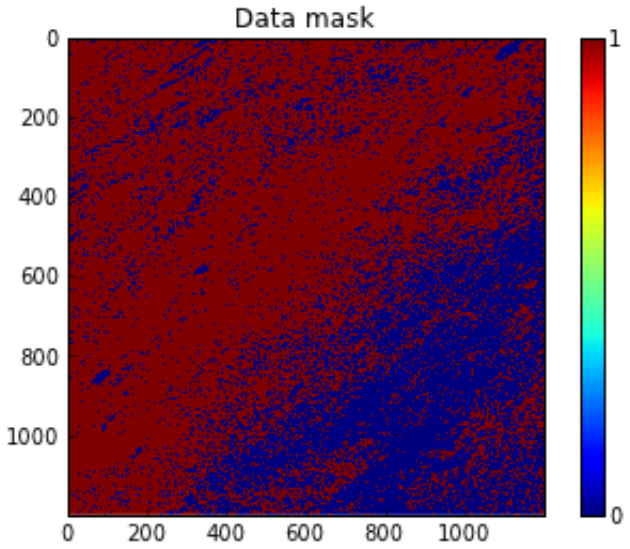


So in the example above, `fapar` is a standard numpy array, so we can use the numpy library to process it, and matplotlib to easily do plots, as done above

In[22]:

```
import numpy as np
import pylab as plt
g = gdal.Open ( 'HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2:FparLai_QC'
qc = g.ReadAsArray()
mask = (qc*0).astype(bool)
okvalues = [0,2]
for i in okvalues:
    mask[np.where(qc == i)] = True


fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.imshow(mask, interpolation='nearest')
cbar = fig.colorbar(cax, ticks=[0,1])
ax.set_title('Data mask')
```

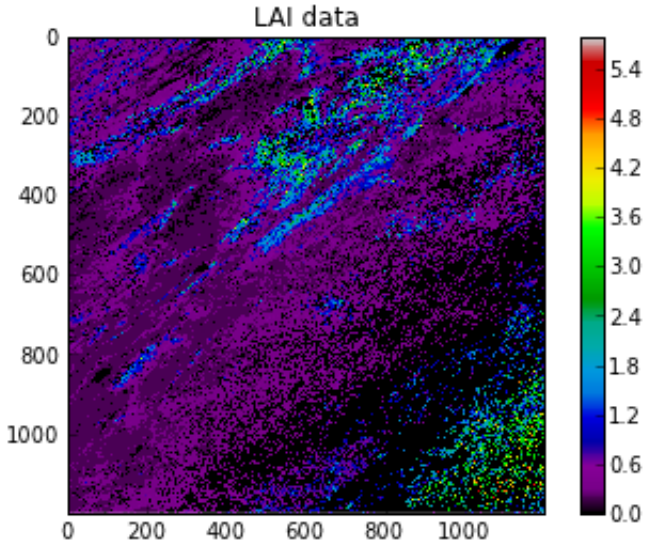
```
plt.show()
```



Use this mask to mask the LAI value where we don't have assurance of good quality...

In[24]:

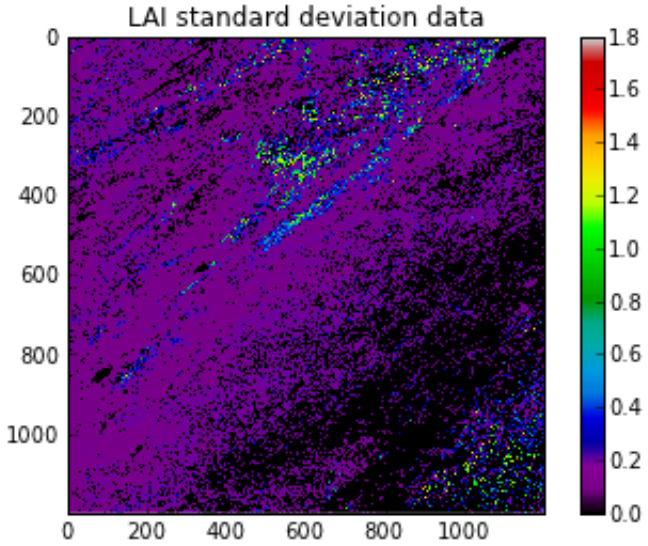
```
g = gdal.Open ( 'HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2:Lai_1km' )
lai = g.ReadAsArray()
fig = plt.figure()
ax = fig.add_subplot(111)
nlai = np.where ( mask == 1, lai*.1, 0.0 )
cax = ax.imshow(nlai, interpolation='nearest', cmap=plt.cm.spectral)
cbar = fig.colorbar(cax)
ax.set_title('LAI data')
plt.show()
```



What about the uncertainty in the estimate of LAI? This is stored as the standard deviation in another layer. We can easily access it, mask it and plot it:

In[27]:

```
g = gdal.Open ( 'HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2:LaiStdDev_1k
sigma_lai = g.ReadAsArray()
fig = plt.figure()
ax = fig.add_subplot(111)
n_sigma_lai = np.where ( mask == 1, sigma_lai*0.1, 0.0 )
cax = ax.imshow(n_sigma_lai, interpolation='nearest', cmap=plt.cm.spectral)
cbar = fig.colorbar(cax)
ax.set_title(r'LAI standard deviation data')
plt.show()
```



More on GDAL

There's quite a bit of extra documentation on using GDAL on http://jgomezdans.github.com/gdal_notes/

Vector datasets

Vector data are usually given in a format called ESRI Shapefile (although other formats also exist). OGR (GDAL's vector sister library) can deal with shapefiles and many other formats.

Getting some data

The file of interest is in the following url: http://txpub.usgs.gov/USACE/data/water_resources/Hydrologic_Units.zip. We will use some shell tools to download this file, but remember that you could use Python for this! However, it's just a one off, so might as well use the shell for this:

```
berlin% mkdir hucData
```

```
berlin% cd hucData
berlin% wget http://txpub.usgs.gov/USACE/data/water_resources/Hydrologic_Units.zip
berlin% curl http://txpub.usgs.gov/USACE/data/water_resources/Hydrologic_Units.zip -o Hydrologic_Units.zip
berlin% unzip Hydrologic_Units.zip
    Archive:  Hydrologic_Units.zip
    ...
    inflating: Hydrologic_Units/HUC_Polygons.shp
```

We can then use `ogrinfo`, the equivalent of `gdalinfo` for vector data, and point to the file with the `.shp` extension. Remember to put the right path to the file, and then to specify the *layer*, which in the case of a Shapefile, it's just the filename without the `.shp` extension. Also, maybe use `less` to page the output

```
ogrinfo Hydrologic_Units/HUC_Polygons.shp HUC_Polygons | less
INFO: Open of `Hydrologic_Units/HUC_Polygons.shp'
  using driver `ESRI Shapefile' successful.

Layer name: HUC_Polygons
Geometry: Polygon
Feature Count: 71
Extent: (-1207861.193700, -1295788.385400) - (-115932.919500, 152769.254400)
Layer SRS WKT:
PROJCS["USA_Contiguous_Albers_Equal_Area_Conic",
    GEOGCS["GCS_North_American_1983",
        DATUM["North_American_Datum_1983",
            SPHEROID["GRS_1980",6378137.0,298.257222101]],
        PRIMEM["Greenwich",0.0],
        UNIT["Degree",0.0174532925199433]],
    PROJECTION["Albers_Conic_Equal_Area"],
    PARAMETER["False_Easting",0.0],
    PARAMETER["False_Northing",0.0],
    PARAMETER["longitude_of_center",-96.0],
    PARAMETER["Standard_Parallel_1",29.5],
    PARAMETER["Standard_Parallel_2",45.5],
    PARAMETER["latitude_of_center",37.5],
    UNIT["Meter",1.0]]
HUC: Integer (9.0)
REG_NAME: String (50.0)
SUB_NAME: String (51.0)
ACC_NAME: String (36.0)
CAT_NAME: String (60.0)
HUC2: Integer (4.0)
```

```
HUC4: Integer (4.0)
HUC6: Integer (9.0)
REG: Integer (4.0)
SUB: Integer (4.0)
ACC: Integer (9.0)
CAT: Integer (9.0)
CAT_NUM: String (8.0)
Shape_Leng: Real (19.11)
Shape_Area: Real (19.11)
OGRFeature(HUC_Polygons):0
  HUC (Integer) = 13010003
  REG_NAME (String) = Rio Grande Region
  SUB_NAME (String) = Rio Grande Headwaters
  ACC_NAME (String) = Rio Grande Headwaters
  CAT_NAME (String) = San Luis. Colorado.
  HUC2 (Integer) = 13
  HUC4 (Integer) = 1301
  HUC6 (Integer) = 130100
  REG (Integer) = 13
  SUB (Integer) = 1301
  ACC (Integer) = 130100
  CAT (Integer) = 13010003
  CAT_NUM (String) = 13010003
  Shape_Leng (Real) =  382736.41359499999
  Shape_Area (Real) = 4133451308.51999998093
  POLYGON ((-828905.55460000038147 49985.936499999836087,-828498.212600000202656 49096.979100000113249,
  [...]
```

Selecting and transforming features

You can do this programmatically in Python, but for simplicity, we will use the shell today in the session. The notes detail how you can do this in Python, and **you are encouraged to read about this and try it out**

Briefly, the process we'll follow will be:

1. Select the feature of interest (our basin of interest)
2. Re-project the data to the MODIS sinusoida projection
3. Build a mask using the MODIS LAI/fAPAR dataset as a reference

Let's say we want to select a feature identified by the HUC code (see output from `ogrinfo`) 13010001 (Rio Grande Headwaters. Colorado). We can use the `ogr2ogr` tool to extract this feature, and reproject.

In GDAL/OGR (and in many other places), projections are defined by WKT (Well-known text) or Proj4. We can go to spatialreference.org and have a look for WKT/Proj4 string that GDAL/OGR need for the projection. In our case, we shall be using the MODIS projection, with a proj4 string given by

```
"+proj=sinu +R=6371007.181 +nadgrids=@null +wktext"
```

`ogr2ogr` requires two options:

1. An option to select features where the field `HUC` is 13010001. For this we'll use the `-where` option.
2. A target (or output) projection definition. This is achieved with the `-t_srs` option

We also need: an output directory (`HUC_Polygon_MODIS`, say), and the input shapefile and the layer name (shapefile name minus `.shp` extension). We build the command line as follows. note that the order of the output files/layers **IS** important!

```
$ ogr2ogr \
          -where "HUC=13010001"\ # Select features where HUC = 13010001
           -t_srs "+proj=sinu +R=6371007.181 +nadgrids=@null +wktext" \ # The output projection
           HUC_Polygons_MODIS \ # The output directory/file
           Hydrologic_Units/HUC_Polygons.shp \ # The input shapefile
           HUC_Polygons # The input shapefile's layer
```

Now test the output in `HUC_Polygons_MODIS/HUC_Polygons.shp` with `ogrinfo` again:

```
$ ogrinfo -al -so HUC_Polygons_MODIS/HUC_Polygons.shp # Note the extra options!
INFO: Open of `/tmp/stuffo/HUC_Polygons.shp'
  using driver `ESRI Shapefile' successful.

Layer name: HUC_Polygons
Geometry: Polygon
Feature Count: 1
Extent: (-9464744.171953, 4160060.958514) - (-9351954.997604, 4221926.101067)
Layer SRS WKT:
```

```
PROJCS["Sinusoidal",
    GEOGCS["GCS_unnamed ellipse",
        DATUM["unknown",
            SPHEROID["Unknown",6371007.181,0]],
    PRIMEM["Greenwich",0],
        UNIT["Degree",0.017453292519943295]],
    PROJECTION["Sinusoidal"],
    PARAMETER["longitude_of_center",0],
    PARAMETER["false_easting",0],
    PARAMETER["false_northing",0],
    UNIT["Meter",1]]
HUC: Integer (9.0)
REG_NAME: String (50.0)
SUB_NAME: String (51.0)
ACC_NAME: String (36.0)
CAT_NAME: String (60.0)
HUC2: Integer (4.0)
HUC4: Integer (4.0)
HUC6: Integer (9.0)
REG: Integer (4.0)
SUB: Integer (4.0)
ACC: Integer (9.0)
CAT: Integer (9.0)
CAT_NUM: String (8.0)
Shape_Leng: Real (19.11)
Shape_Area: Real (19.11)
```

Compare the projection reported now by `ogrinfo` with that reported earlier by `gdalinfo`. Also see how the reported extenst for both datasets relate.

Create a KML and upload it to Google Maps...

If you specify

- KML as your output format
- EPSG:4326 latitude/longitude reference system

you can create a KML file suitable for viewing in Google Earth/Maps

```
ogr2ogr -f "KML" \ # Output set to KML format
         -where "HUC=13010001" # Select feature
         -t_srs "EPSG:4326"  \ # Output projection
         Hydrologic_Units_MODIS.kml \ # Output filename
         Hydrologic_Units/HUC_Polygons.shp \ # As above
         HUC_Polygons # As above
```

Creating the mask

We shall use `gdal_rasterize` to create the mask. We'll create a GeoTIFF file. What we need

- The target projection
- The output format
- The extent of the output file
- The number of rows & columns in the output file
- The datatype

We can gleam this information from the MODIS HDF file using `gdalinfo` and the commentws above. This is the command to use:

```
gdal_rasterize -burn 1 \
    -of GTiff \
    -a_srs "+proj=sinu +R=6371007.181 +nadgrids=@null +wktext" \
    -te -10007554.677 3335851.559 -8895604.157 4447802.079 \
    -ts 1200 1200 \
    -ot Byte \
    Hydrologic_Units_MODIS/HUC_Polygons.shp \
    HUC_mask.tif
```

Check that `HUC_mask.tif` is indeed GDAL-readable by using `gdalinfo` as before.

Using the mask

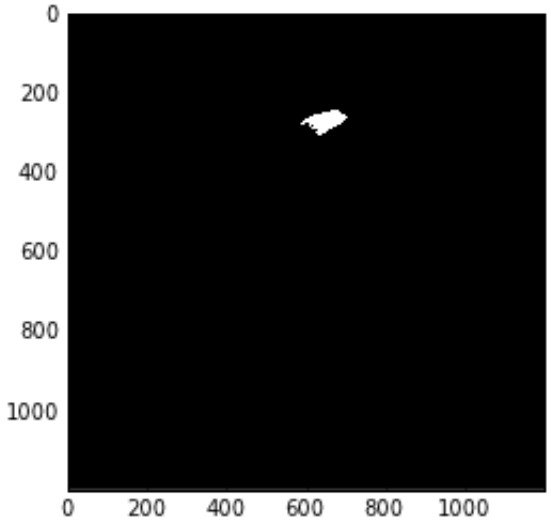
Let's read the mask the some LAI data...

In[30]:

```
from osgeo import gdal
import numpy as np
import matplotlib.pyplot as plt

g = gdal.Open( 'HDF4_EOS:EOS_GRID:"MCD15A2.A2011193.h09v05.005.2011203175347.hdf":MOD_Grid_MOD15A2:Lai_1km')
lai = g.ReadAsArray()*0.1

g = gdal.Open ( "HUC_mask.tif" )
mask = g.ReadAsArray()

plt.imshow ( mask, interpolation='nearest', cmap=plt.cm.gray )
```

Out[30]:

```
<matplotlib.image.AxesImage at 0xae0042c>
```

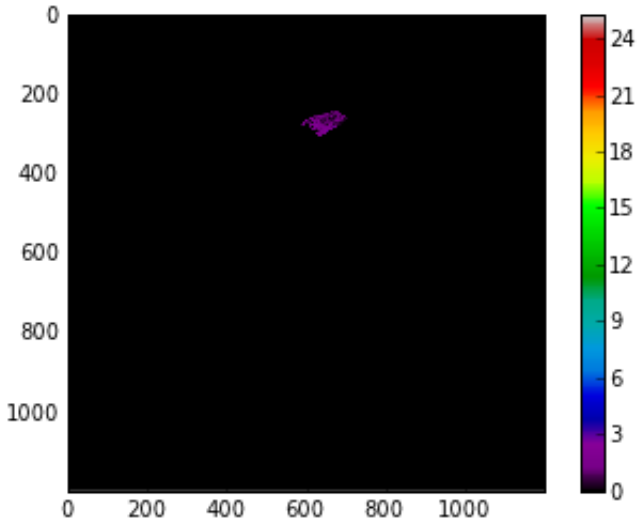


In[38]:

```
# We can use the mask to select MODIS data from the raster
lai_m = np.where ( mask==1, lai, 0 )
plt.imshow ( lai_m, interpolation='nearest', cmap=plt.cm.spectral )
plt.colorbar()
plt.figure()
# Why does it go up to 24?
```

Out[38]:

```
<matplotlib.figure.Figure at 0xb728ccc>
```



In[39]:

```
# The mean value of LAI within the catchment...
mean_lai = lai_m[lai_m > 0].mean()
std_lai = lai_m[lai_m > 0].std()
print mean_lai, std_lai
```

```
1.33370674633 1.90879912294
```