GIS 4653/5653: Spatial Programming and GIS

Basic GIS

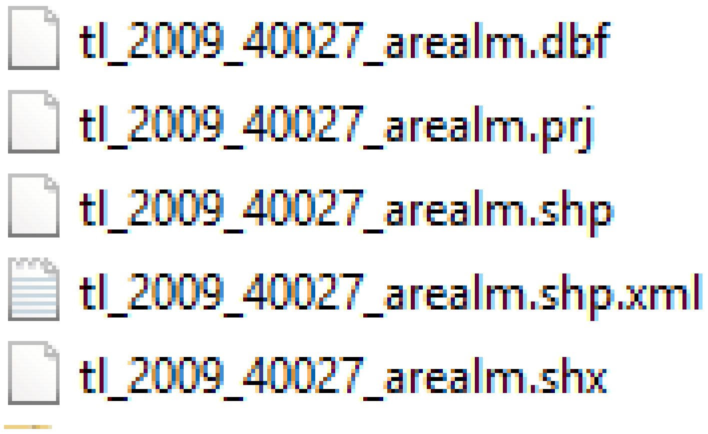
Reading and writing shapefiles

GIS datasets

- GIS datasets tend to come in some quasi-standard format
  - Open-source Python modules available to read these
  - Can then process the data in Python
- Examples of freely available GIS datasets
  - TIGER (census.gov): county information in Shapefile format
  - naturalearthdata.com: borders, timezones, roads, etc. in Shapefile format

What are Shapefiles?

- Shapefiles are a GIS data format
  - Originated by ESRI
  - The specification is open, so many modules exist to read/write shapefiles
- A shapefile is not a single file, but instead a group of files



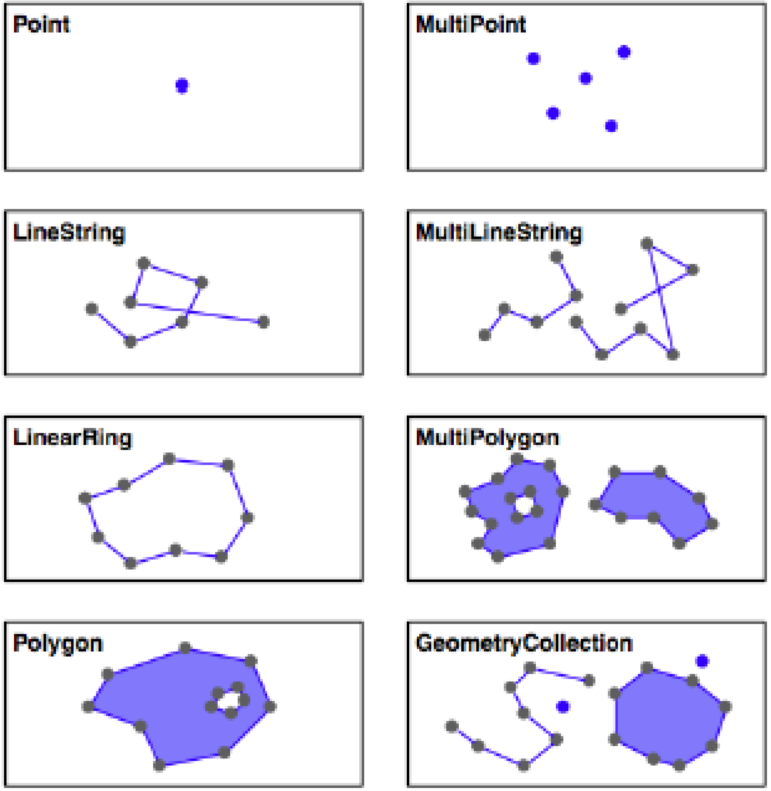
tl_2009_40027_arealm.dbf
tl_2009_40027_arealm.prj
tl_2009_40027_arealm.shp
tl_2009_40027_arealm.shp.xml
tl_2009_40027_arealm.shx

The key three components

- Three key components:
  - The .shp file contains the geometry
  - The .dbf file contains the attributes as a relational table
  - The .prj file contains the map projection as well-known text

Types of shapes

- Shapefiles are typically composed of a number of shapes of a single type.



Source: Python Geospatial Development by Erik Westra, 2010

Census Tiger Files

- Download census data for Cleveland County:
  - http://www2.census.gov/cgi-bin/shapefiles2009/county-files?county=40027
  - Includes TGRSH09.pdf which has details about the shapefiles

Displaying in ArcMap

- Areawater and block-group 00

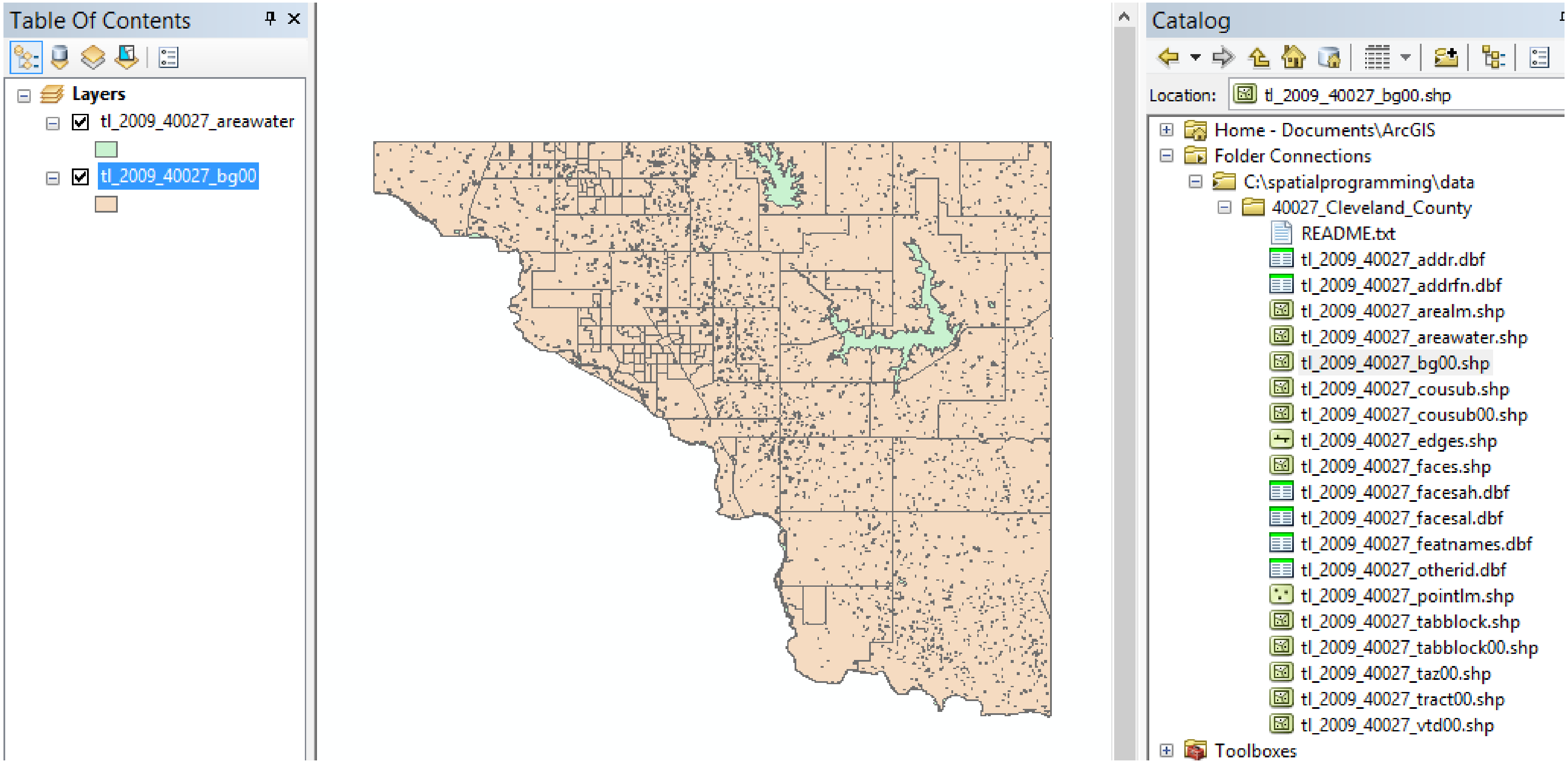


Table Of Contents

- Layers
  - ☑ tl_2009_40027_areawater
    - 🟩
  - ☑ tl_2009_40027_bg00
    - 🟧

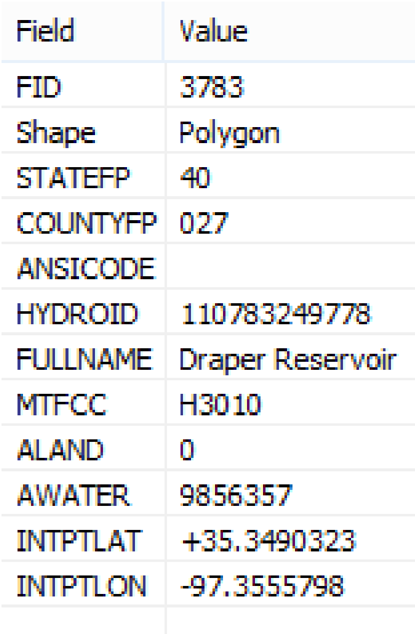
Catalog

Location: tl_2009_40027_bg00.shp

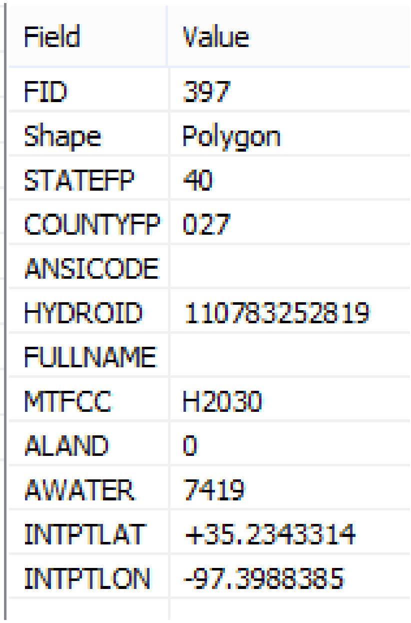
- Home - Documents\ArcGIS
- Folder Connections
  - C:\spatialprogramming\data
    - 40027_Cleveland_County
      - README.txt
      - tl_2009_40027_addr.dbf
      - tl_2009_40027_addrfn.dbf
      - tl_2009_40027_arealm.shp
      - tl_2009_40027_areawater.shp
      - tl_2009_40027_bg00.shp
      - tl_2009_40027_cousub.shp
      - tl_2009_40027_cousub00.shp
      - tl_2009_40027_edges.shp
      - tl_2009_40027_faces.shp
      - tl_2009_40027_facesah.dbf
      - tl_2009_40027_facesal.dbf
      - tl_2009_40027_featnames.dbf
      - tl_2009_40027_otherid.dbf
      - tl_2009_40027_pointlm.shp
      - tl_2009_40027_tabblock.shp
      - tl_2009_40027_tabblock00.shp
      - tl_2009_40027_taz00.shp
      - tl_2009_40027_tract00.shp
      - tl_2009_40027_vtd00.shp
- Toolboxes

Problem …

- Not all water bodies in the shapefile have names
  - We would like to extract out water bodies that have names and write out a separate shapefile



Field	Value
FID	3783
Shape	Polygon
STATEFP	40
COUNTYFP	027
ANSICODE	
HYDROID	110783249778
FULLNAME	Draper Reservoir
MTFCC	H3010
ALAND	0
AWATER	9856357
INTPTLAT	+35.3490323
INTPTLON	-97.3555798

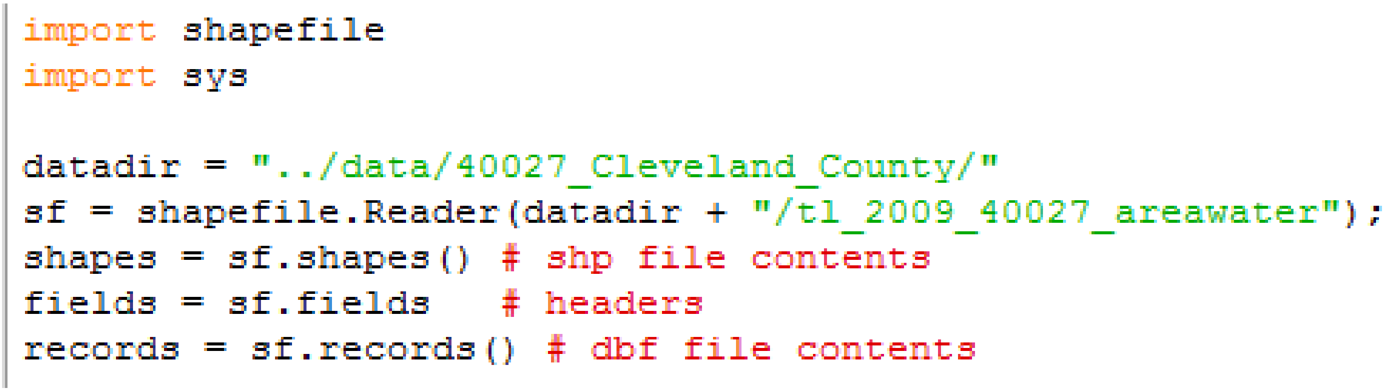


Field	Value
FID	397
Shape	Polygon
STATEFP	40
COUNTYFP	027
ANSICODE	
HYDROID	110783252819
FULLNAME	
MTFCC	H2030
ALAND	0
AWATER	7419
INTPTLAT	+35.2343314
INTPTLON	-97.3988385

- How would you go about this?
  - Do this for every county in US …

Reading shapefiles in Python

- One of the ways to read shapefiles is use PyShp
  - https://code.google.com/p/pyshp/
  - Download and place shapefile.py along with the rest of your code

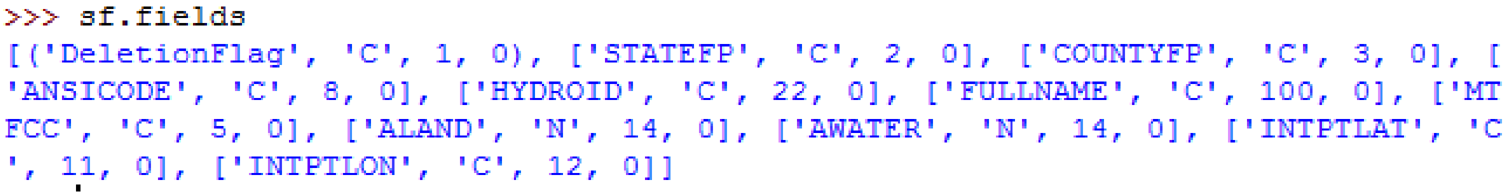


```
import shapefile
import sys

datadir = "../data/40027_Cleveland_County/"
sf = shapefile.Reader(datadir + "/tl_2009_40027_areawater");
shapes = sf.shapes() # shp file contents
fields = sf.fields   # headers
records = sf.records() # dbf file contents
```

- Now what?

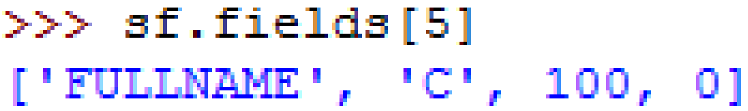
Looking at headers

- Which field number is the FULLNAME field?



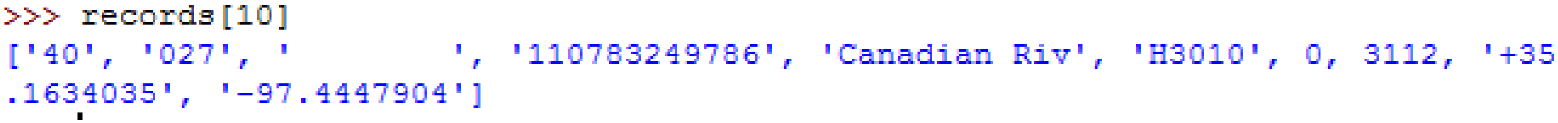
```
>>> sf.fields
[('DeletionFlag', 'C', 1, 0), ['STATEFP', 'C', 2, 0], ['COUNTYFP', 'C', 3, 0], [
'ANSICODE', 'C', 8, 0], ['HYDROID', 'C', 22, 0], ['FULLNAME', 'C', 100, 0], ['MT
FCC', 'C', 5, 0], ['ALAND', 'N', 14, 0], ['AWATER', 'N', 14, 0], ['INTPTLAT', 'C
', 11, 0], ['INTPTLON', 'C', 12, 0]]
```

- Make sure …



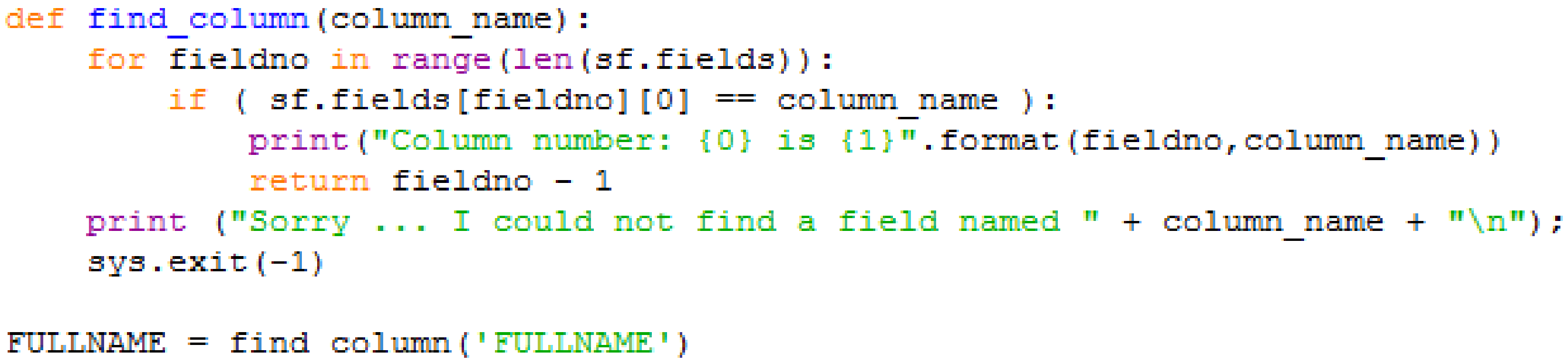
```
>>> sf.fields[5]
['FULLNAME', 'C', 100, 0]
```

- Look at an example record … what field# in record?



```
>>> records[10]
['40', '027', '        ', '110783249786', 'Canadian Riv', 'H3010', 0, 3112, '+35
.1634035', '-97.4447904']
```

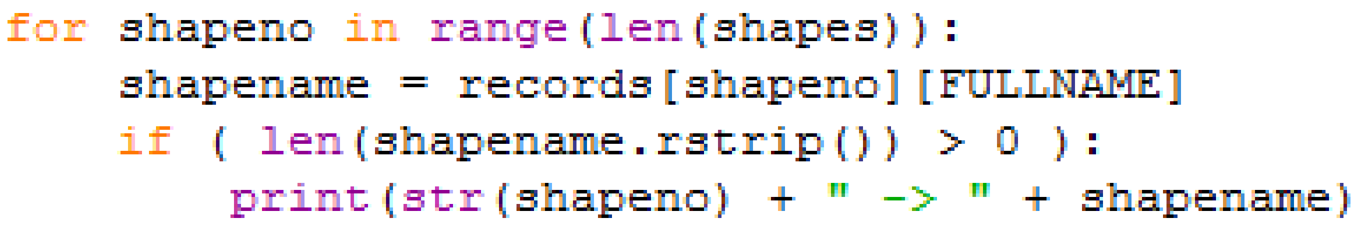
Can do this programmatically



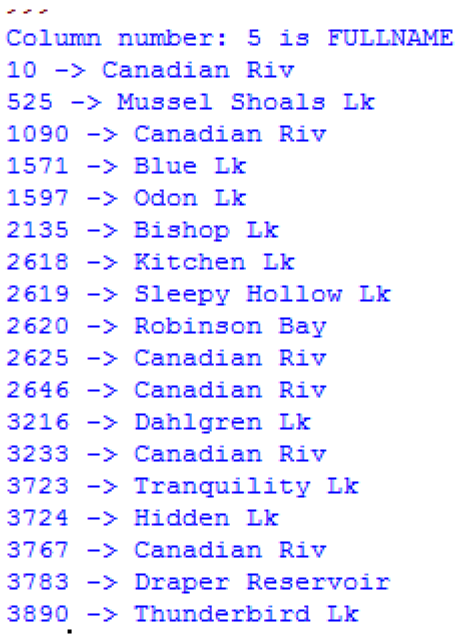
```
def find_column(column_name):
    for fieldno in range(len(sf.fields)):
        if ( sf.fields[fieldno][0] == column_name ):
            print("Column number: {0} is {1}".format(fieldno,column_name))
            return fieldno - 1
    print ("Sorry ... I could not find a field named " + column_name + "\n");
    sys.exit(-1)

FULLNAME = find column('FULLNAME')
```

- Why is this approach better?
- Now what?

Finding shapes with names



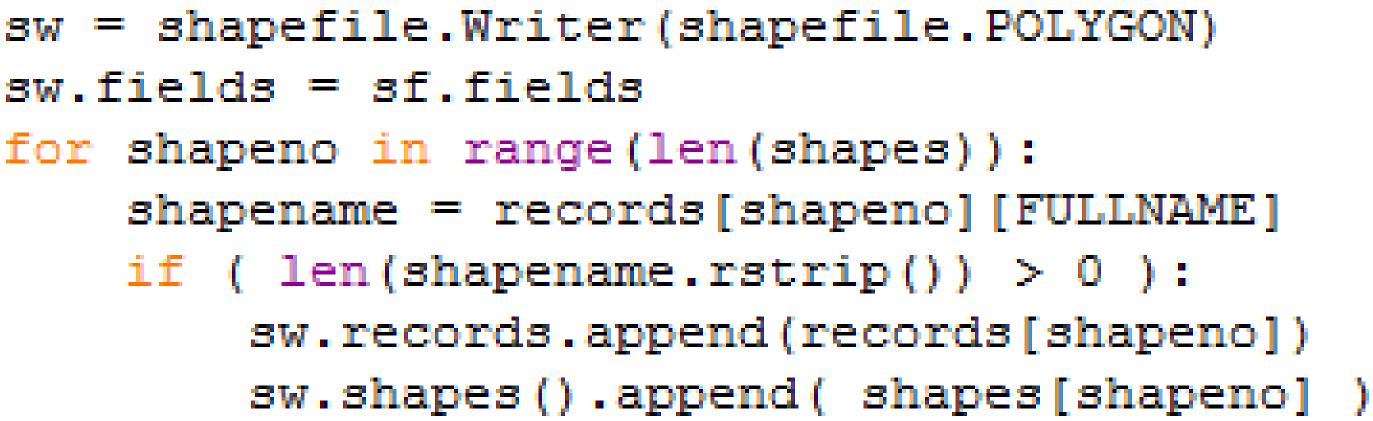
```python
for shapeno in range(len(shapes)):
    shapename = records[shapeno][FULLNAME]
    if ( len(shapename.rstrip()) > 0 ):
        print(str(shapeno) + " -> " + shapename)
```



```
...
Column number: 5 is FULLNAME
10 -> Canadian Riv
525 -> Mussel Shoals Lk
1090 -> Canadian Riv
1571 -> Blue Lk
1597 -> Odon Lk
2135 -> Bishop Lk
2618 -> Kitchen Lk
2619 -> Sleepy Hollow Lk
2620 -> Robinson Bay
2625 -> Canadian Riv
2646 -> Canadian Riv
3216 -> Dahlgren Lk
3233 -> Canadian Riv
3723 -> Tranquility Lk
3724 -> Hidden Lk
3767 -> Canadian Riv
3783 -> Draper Reservoir
3890 -> Thunderbird Lk
```

Writing out a shapefile

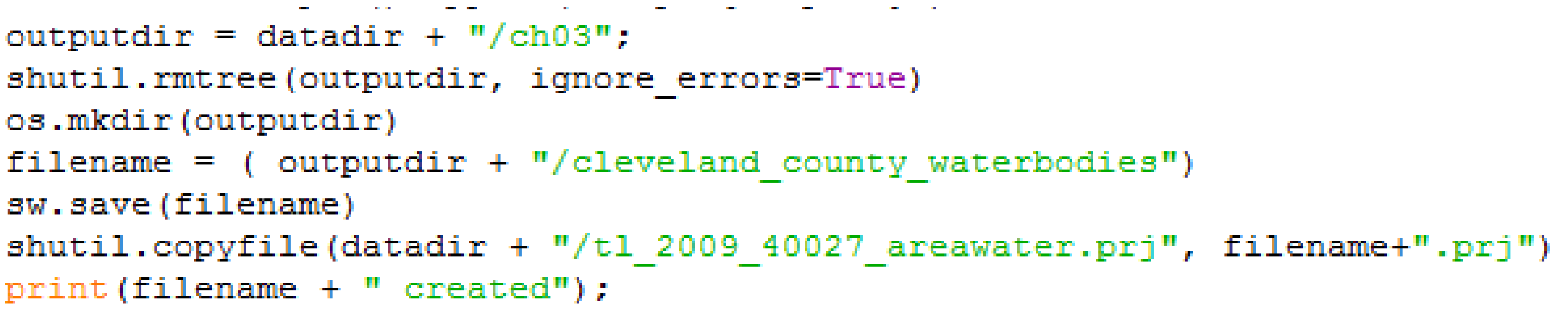
- Set up the shapefile:



```
sw = shapefile.Writer(shapefile.POLYGON)
sw.fields = sf.fields
for shapeno in range(len(shapes)):
    shapename = records[shapeno][FULLNAME]
    if ( len(shapename.rstrip()) > 0 ):
        sw.records.append(records[shapeno])
        sw.shapes().append( shapes[shapeno] )
```

Saving the shapefile

- The shapefile module will not overwrite files
  - So remove the output directory before writing it
  - It does not write a .prj file, so copy original .prj file …

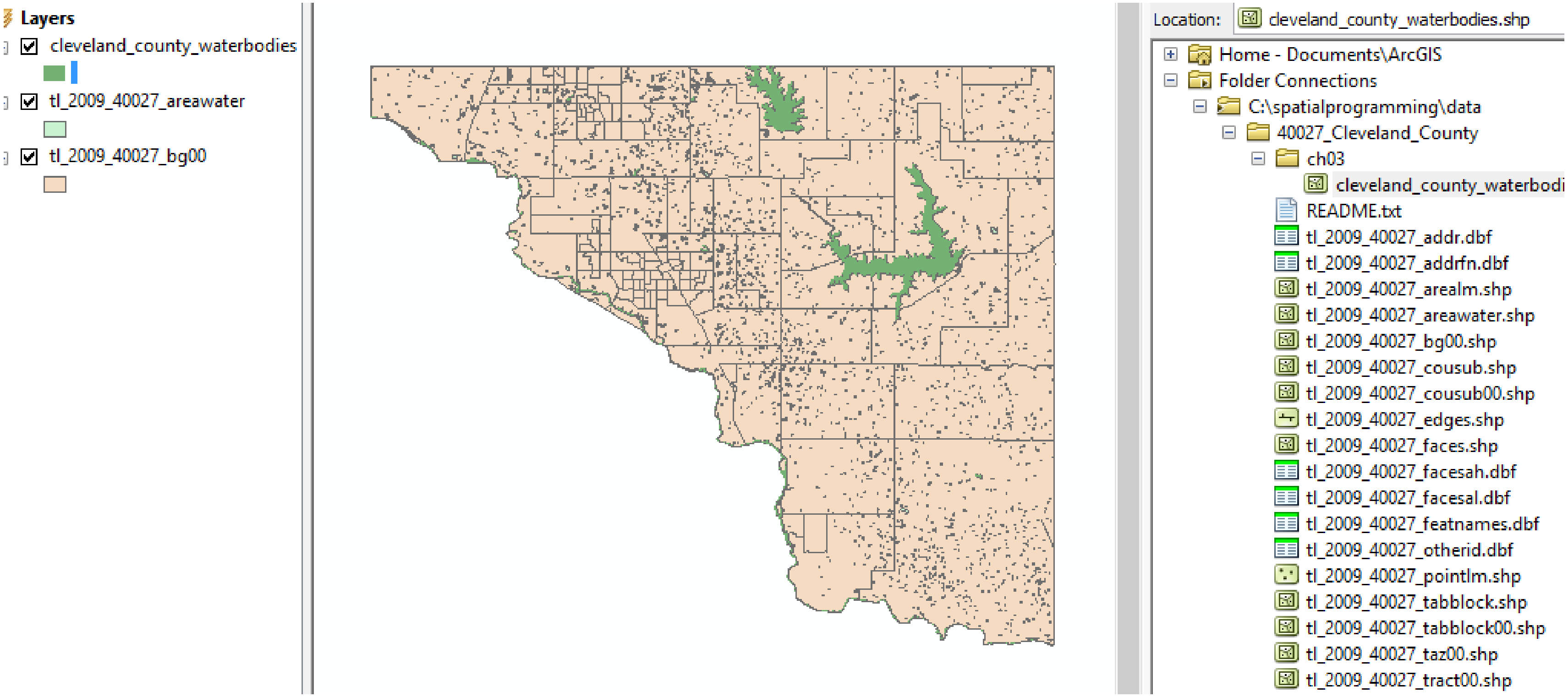


```
outputdir = datadir + "/ch03";
shutil.rmtree(outputdir, ignore_errors=True)
os.mkdir(outputdir)
filename = ( outputdir + "/cleveland_county_waterbodies")
sw.save(filename)
shutil.copyfile(datadir + "/tl_2009_40027_areawater.prj", filename+".prj")
print(filename + " created");
```

Displaying Shapefiles

Displaying result in ArcMap

- The full code is at waterbodies.py



Layers

☑ cleveland_county_waterbodies

🟩

☑ tl_2009_40027_areawater

🟩

☑ tl_2009_40027_bg00

🟧

Location: cleveland_county_waterbodies.shp

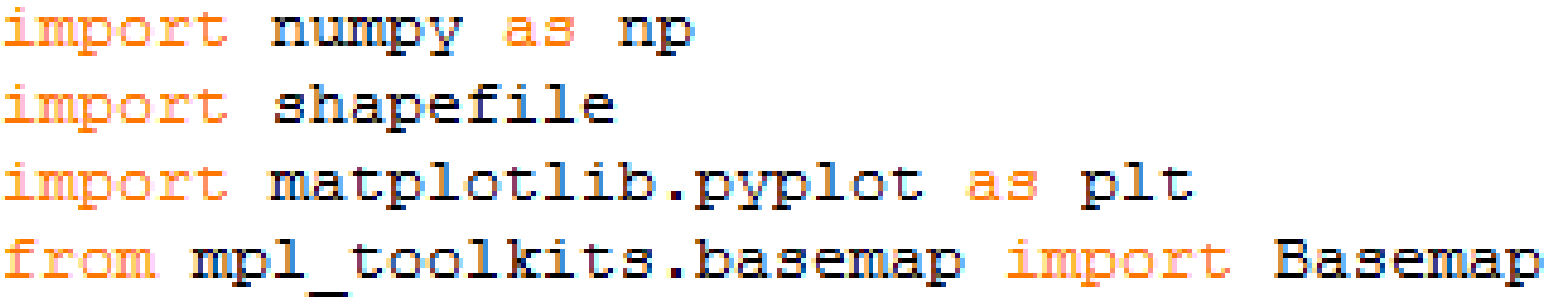
- Home - Documents\ArcGIS
- Folder Connections
  - C:\spatialprogramming\data
    - 40027_Cleveland_County
      - ch03
        - cleveland_county_waterbodi
      - README.txt
      - tl_2009_40027_addr.dbf
      - tl_2009_40027_addrfn.dbf
      - tl_2009_40027_arealm.shp
      - tl_2009_40027_areawater.shp
      - tl_2009_40027_bg00.shp
      - tl_2009_40027_cousub.shp
      - tl_2009_40027_cousub00.shp
      - tl_2009_40027_edges.shp
      - tl_2009_40027_faces.shp
      - tl_2009_40027_facesah.dbf
      - tl_2009_40027_facesal.dbf
      - tl_2009_40027_featnames.dbf
      - tl_2009_40027_otherid.dbf
      - tl_2009_40027_pointlm.shp
      - tl_2009_40027_tabblock.shp
      - tl_2009_40027_tabblock00.shp
      - tl_2009_40027_taz00.shp
      - tl_2009_40027_tract00.shp

Displaying without ArcMap

- Useful to be able to display GIS data without ArcMap
  - Useful for dynamically creating maps
  - License/cost issues
- Can use a combination of modules to display GIS data
  - numpy for numeric processing
  - matplotlib for plotting
  - Basemap for projections and reading shapefiles into displayable points

Module imports

- First import the necessary modules
  - Using common aliases



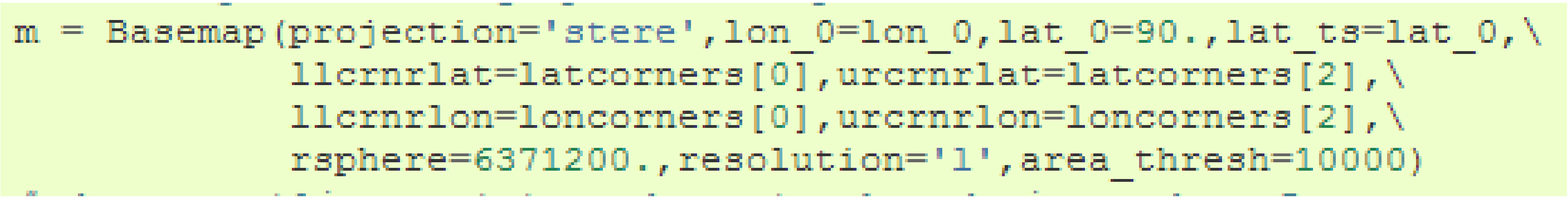
```
import numpy as np
import shapefile
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
```

Drawing maps with Python

- The process of creating and drawing maps:
  - Create a Basemap
  - Read shapefile
  - Plot the shapes in the shapefile
  - Show the plot

Creating a Basemap

- To create a Basemap, specify the bounds of the plot:

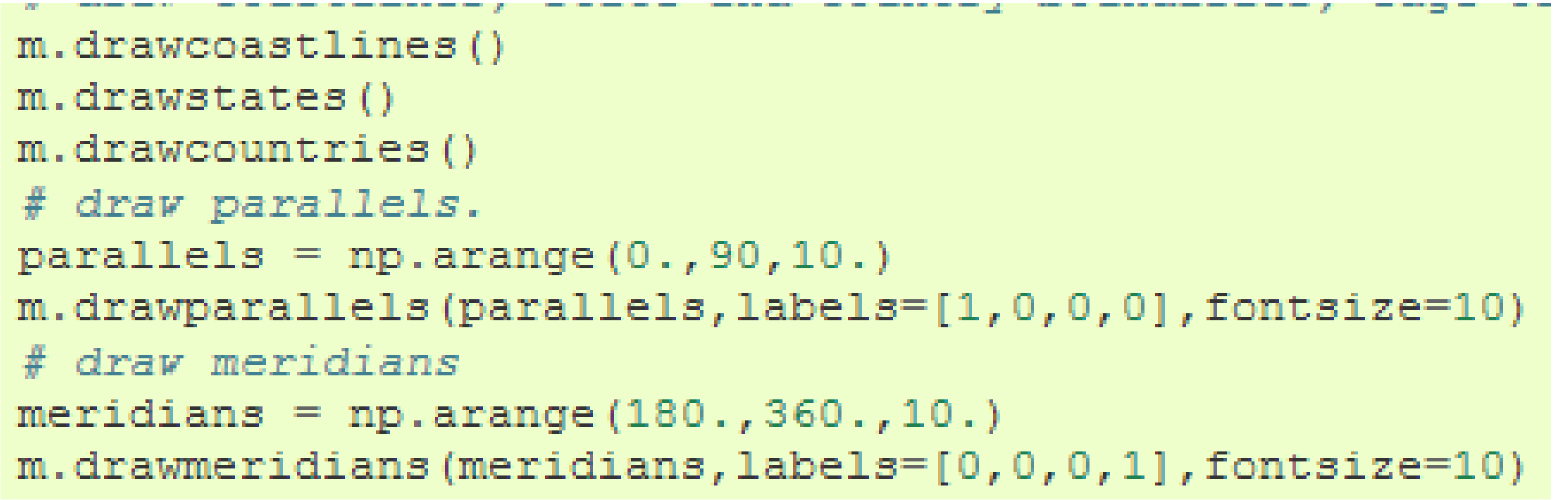


```
m = Basemap(projection='stere',lon_0=lon_0,lat_0=90.,lat_ts=lat_0,\
            llcrnrlat=latcorners[0],urcrnrlat=latcorners[2],\
            llcrnrlon=loncorners[0],urcrnrlon=loncorners[2],\
            rsphere=6371200.,resolution='l',area_thresh=10000)
```

- Can choose from several projections: stereographic, Mercatur, Robinson, Lambert Conformal, etc.
  - Specify any necessary parameters for the projection
- Specify bounding box
  - How fine/coarse do you want the drawing to be?
  - Areas smaller than what should be ignored?

Built-in maps

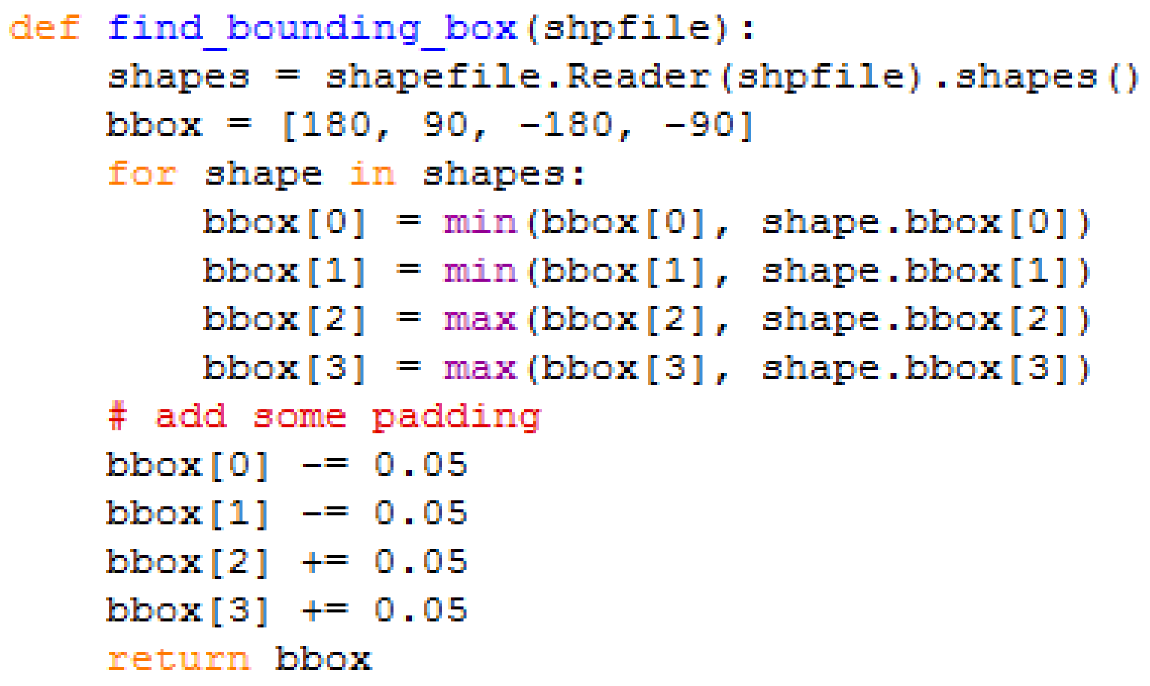
- Some basic map features are built-in
  - You don’t need extra shapefiles for these:



```
m.drawcoastlines()
m.drawstates()
m.drawcountries()
# draw parallels.
parallels = np.arange(0.,90,10.)
m.drawparallels(parallels,labels=[1,0,0,0],fontsize=10)
# draw meridians
meridians = np.arange(180.,360.,10.)
m.drawmeridians(meridians,labels=[0,0,0,1],fontsize=10)
```

Bounding box

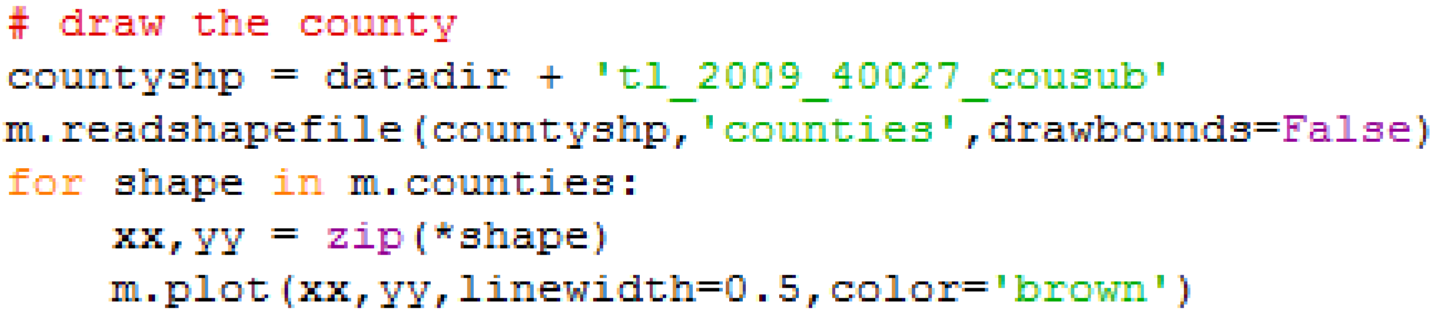
- How do you find the bounding box of a shapefile?
  - Can use the shapefile module to read shapes and compute this



```
def find_bounding_box(shpfile):
    shapes = shapefile.Reader(shpfile).shapes()
    bbox = [180, 90, -180, -90]
    for shape in shapes:
        bbox[0] = min(bbox[0], shape.bbox[0])
        bbox[1] = min(bbox[1], shape.bbox[1])
        bbox[2] = max(bbox[2], shape.bbox[2])
        bbox[3] = max(bbox[3], shape.bbox[3])
    # add some padding
    bbox[0] -= 0.05
    bbox[1] -= 0.05
    bbox[2] += 0.05
    bbox[3] += 0.05
    return bbox
```

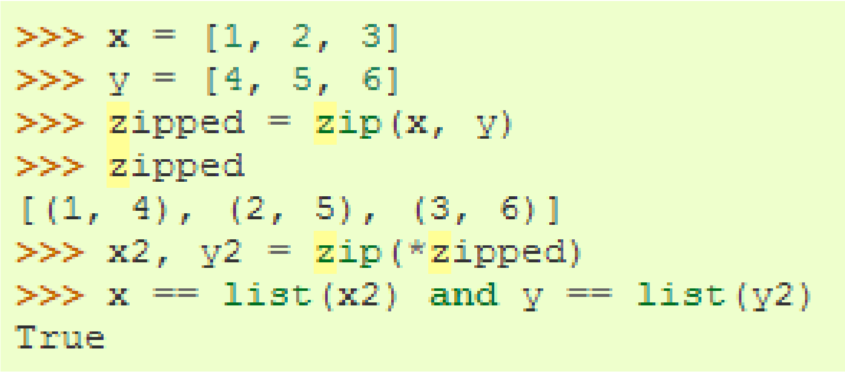
Reading and plotting a shapefile

- Reading a shapefile returns a list of tuples (list of points)



```
# draw the county
countyshp = datadir + 'tl_2009_40027_cousub'
m.readshapefile(countyshp,'counties',drawbounds=False)
for shape in m.counties:
    xx,yy = zip(*shape)
    m.plot(xx,yy,linewidth=0.5,color='brown')
```

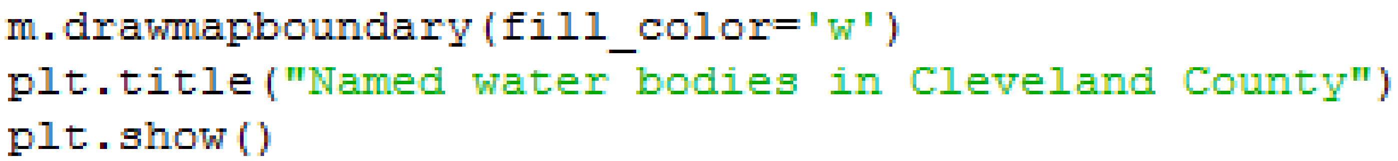
- zip() is a built-in Python function (zip as in fastener)
  - zip() with the * operator essentially unzips



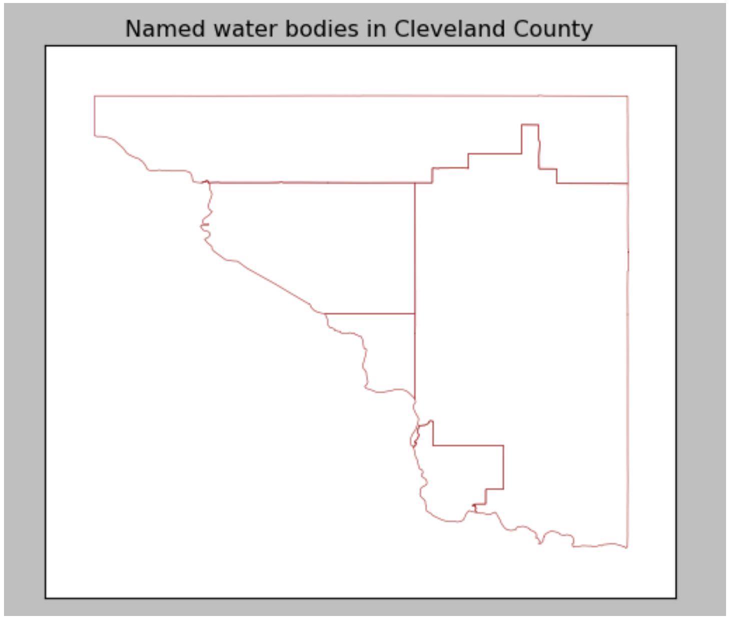
```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> zipped = zip(x, y)
>>> zipped
[(1, 4), (2, 5), (3, 6)]
>>> x2, y2 = zip(*zipped)
>>> x == list(x2) and y == list(y2)
True
```

Setting up plot and drawing it

- The matplotlib is used for plotting
    - Can plot all types of charts and figures

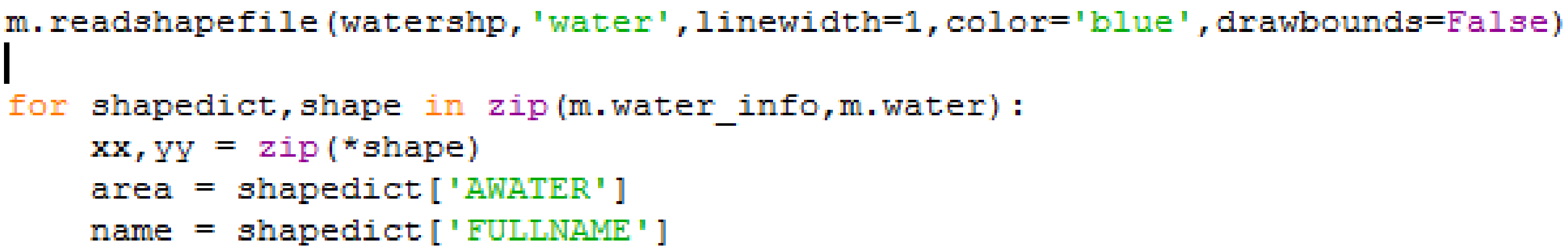


```
m.drawmapboundary(fill_color='w')
plt.title("Named water bodies in Cleveland County")
plt.show()
```



Getting the attributes

- Basemap also reads the attributes
  - Makes the shapes and attributes available
  - For example:

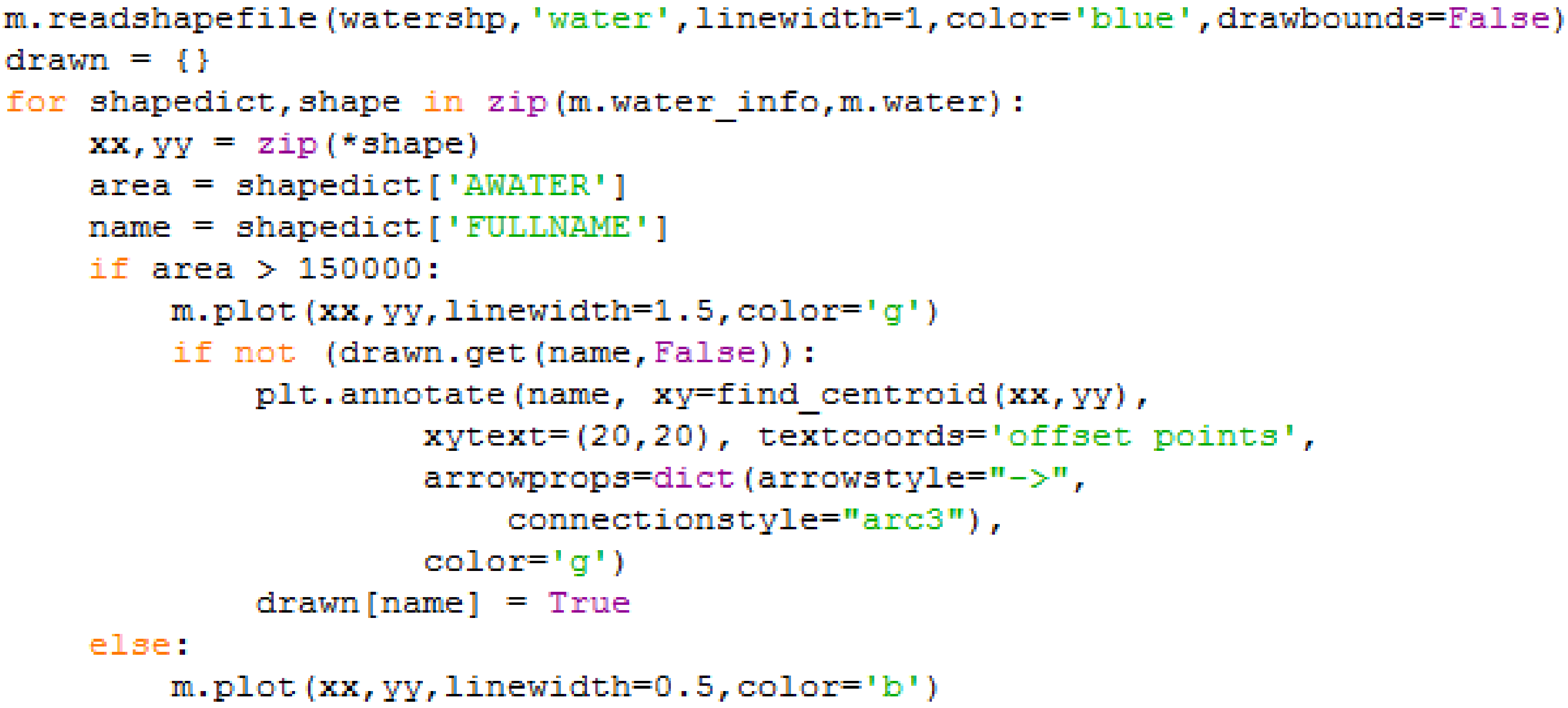


```
m.readshapefile(watershp,'water',linewidth=1,color='blue',drawbounds=False)

for shapedict,shape in zip(m.water_info,m.water):
    xx,yy = zip(*shape)
    area = shapedict['AWATER']
    name = shapedict['FULLNAME']
```

  - The shapes are in ‘water’ and attributes in ‘water_info’ because the second parameter to readshapefile is ‘water’

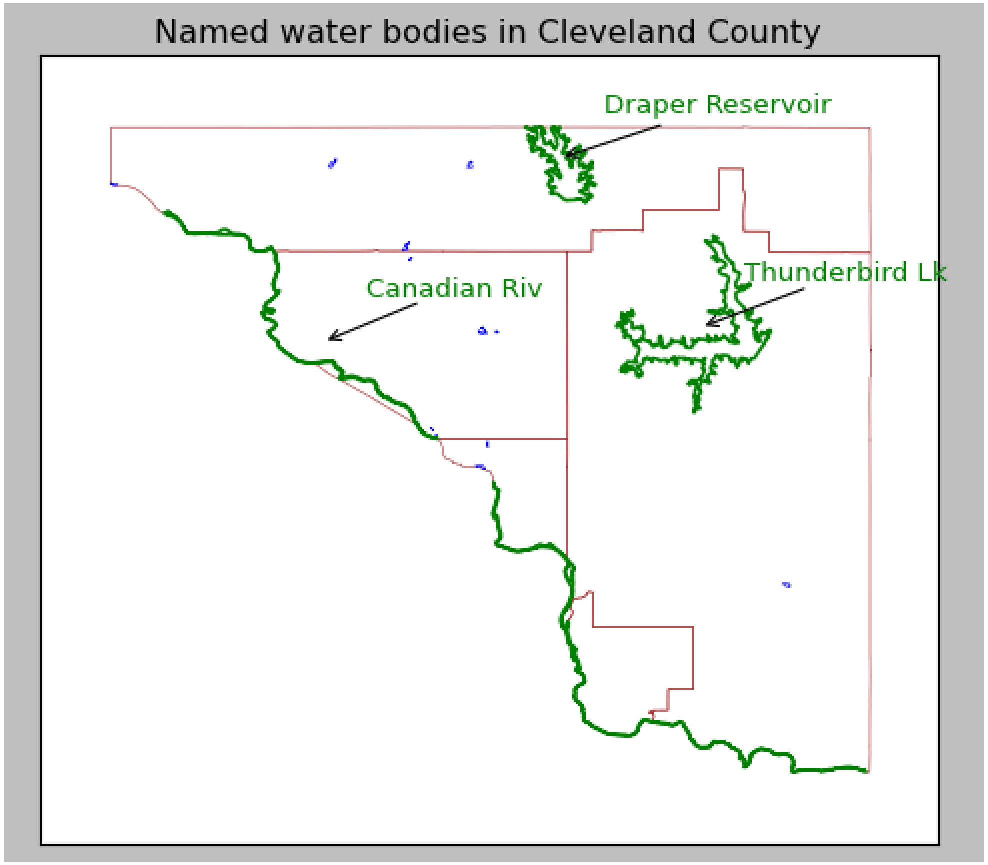
Choosing what to draw

- Can use the attributes to choose what/how to draw
    - Multiple shapes may have the same name (since a Polygon may consist of multiple PolygonRings when being drawn)



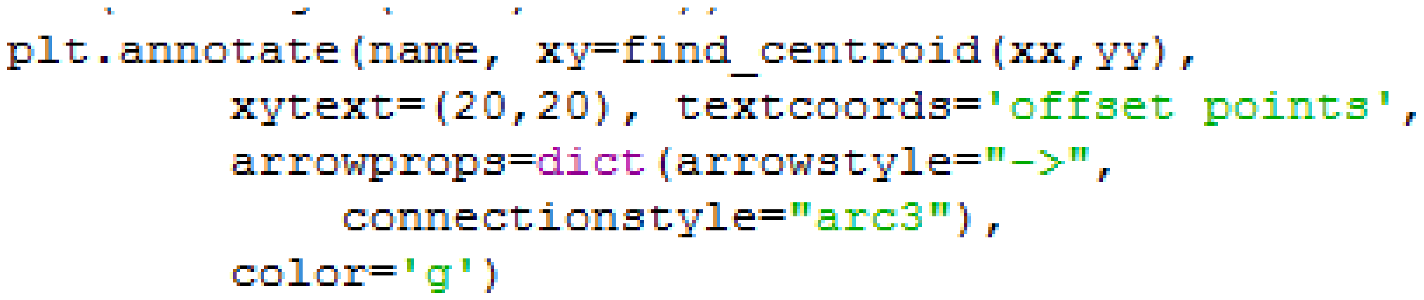
```
m.readshapefile(watershp,'water',linewidth=1,color='blue',drawbounds=False)
drawn = {}
for shapedict,shape in zip(m.water_info,m.water):
    xx,yy = zip(*shape)
    area = shapedict['AWATER']
    name = shapedict['FULLNAME']
    if area > 150000:
        m.plot(xx,yy,linewidth=1.5,color='g')
        if not (drawn.get(name,False)):
            plt.annotate(name, xy=find_centroid(xx,yy),
                    xytext=(20,20), textcoords='offset points',
                    arrowprops=dict(arrowstyle="->",
                        connectionstyle="arc3"),
                    color='g')
            drawn[name] = True
    else:
        m.plot(xx,yy,linewidth=0.5,color='b')
```

Final result



Annotation is done using matplotlib

- Point to a location xy
  - The location is provided in data units (default)
- Place text at a location xytext
  - The location is provided as an offset from xy in figure units
- Draw an arrow between the two points



```
plt.annotate(name, xy=find_centroid(xx,yy),
        xytext=(20,20), textcoords='offset points',
        arrowprops=dict(arrowstyle="->",
            connectionstyle="arc3"),
        color='g')
```

Reference documentation

- Please refer to the documentation of the three modules we have used:
- http://matplotlib.org/api/pyplot_api.html
- https://code.google.com/p/pyshp/
- http://matplotlib.org/basemap/api/basemap_api.html

Homework

- Download Tiger data for Cleveland county
  - Identify water bodies with area larger than 10,000 (look at the AWATER field) that have no name
  - Write out a shapefile of just these water bodies
- Your report (PDF) should consist of:
  - The list of such water bodies
  - Display in ArcMap zoomed in on the largest of these waterbodies
  - Display outside of ArcMap of the entire area
- You can download my example program from the same place you got this PDF: waterbodies.py and showwater.py

Raster data

Raster vs. Vector

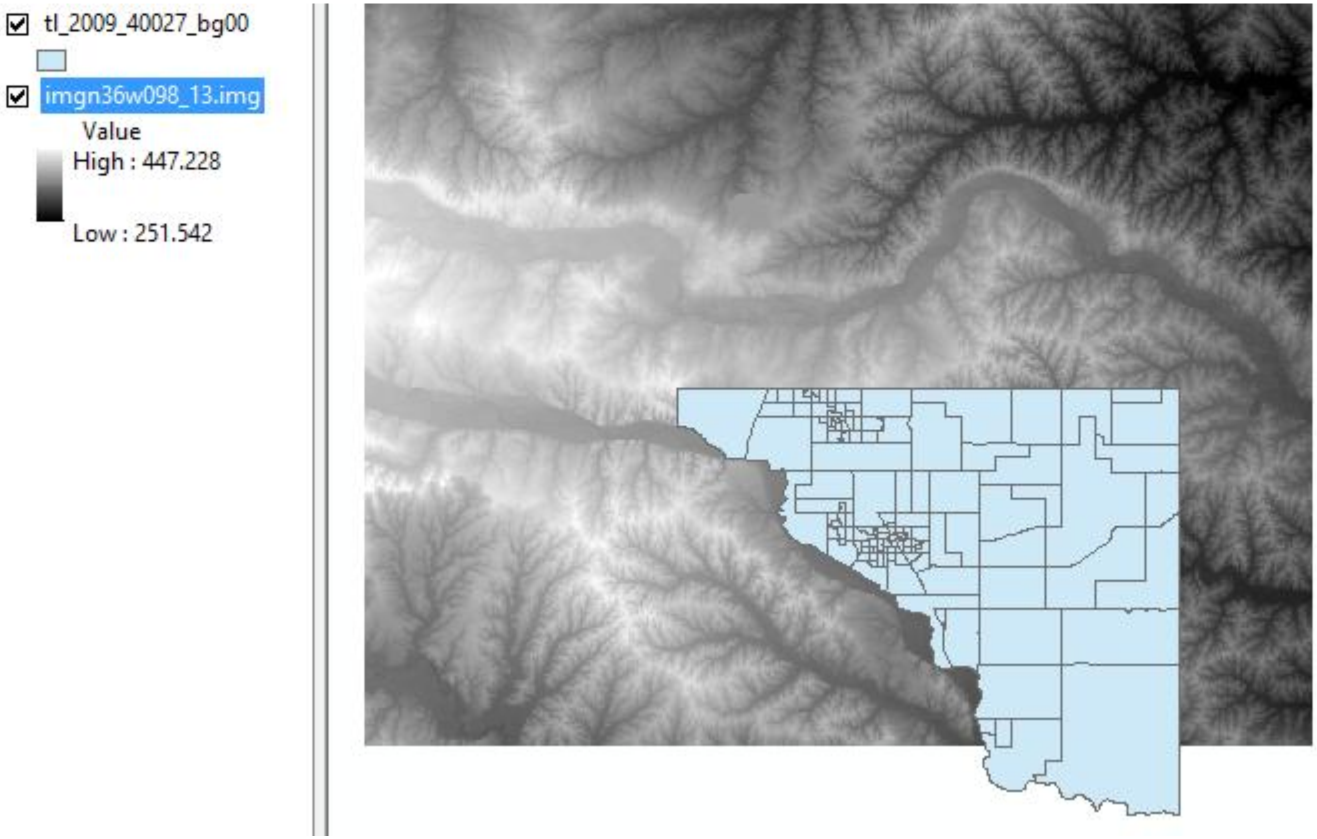
- What’s the difference?

Elevation data

- Can obtain elevation data for the Cleveland county area:
  - http://viewer.nationalmap.gov/viewer/
  - Zoom in and center on Norman, making sure you see Draper Reservoir, Thunderbird Lake and the Canadian River
  - Click on "Download Data"
  - Choose "Click here to order for map extent"
  - Select Elevation and get the National Elevation Dataset (1/3 arc second) in IMG format

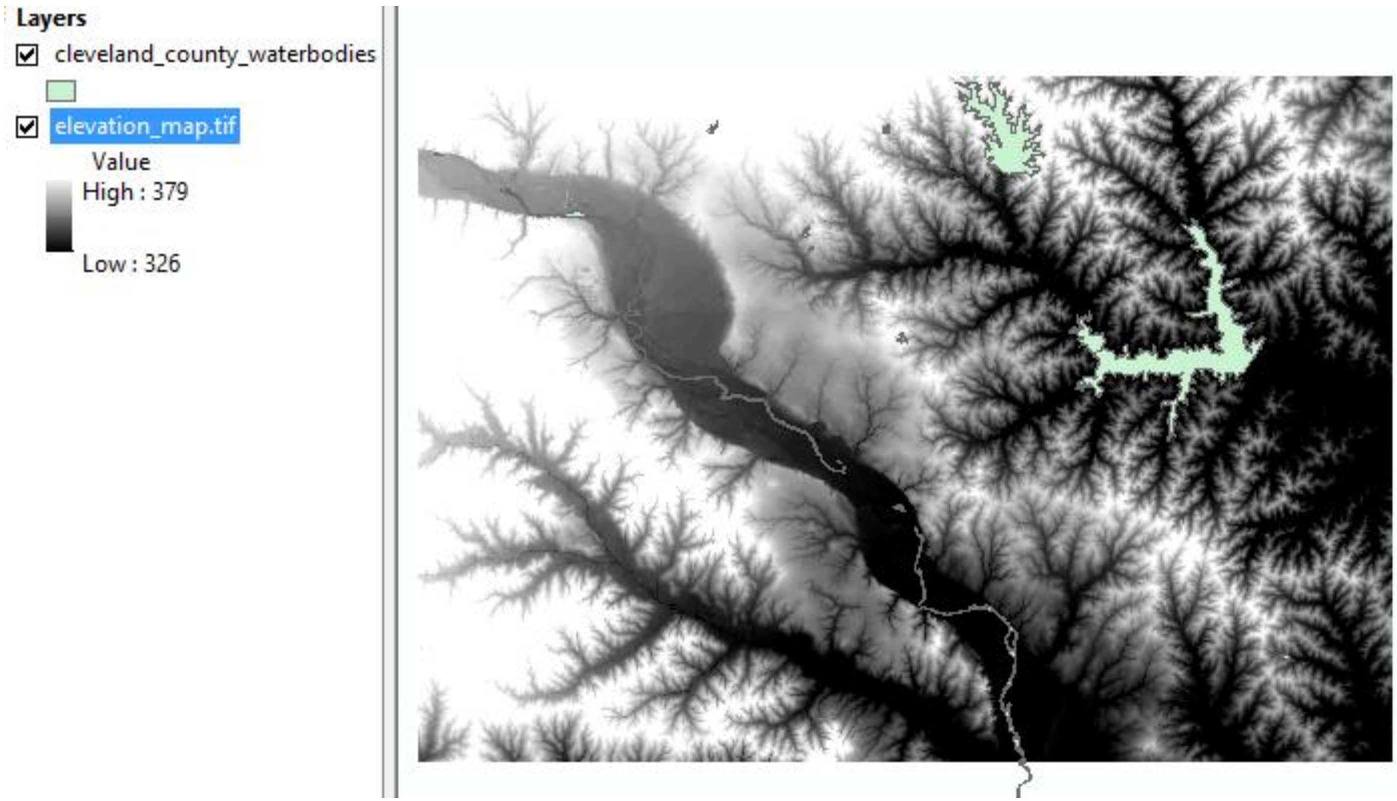
Elevation data in ArcMap

- 350MB … and the county is in two patches … oh, well.



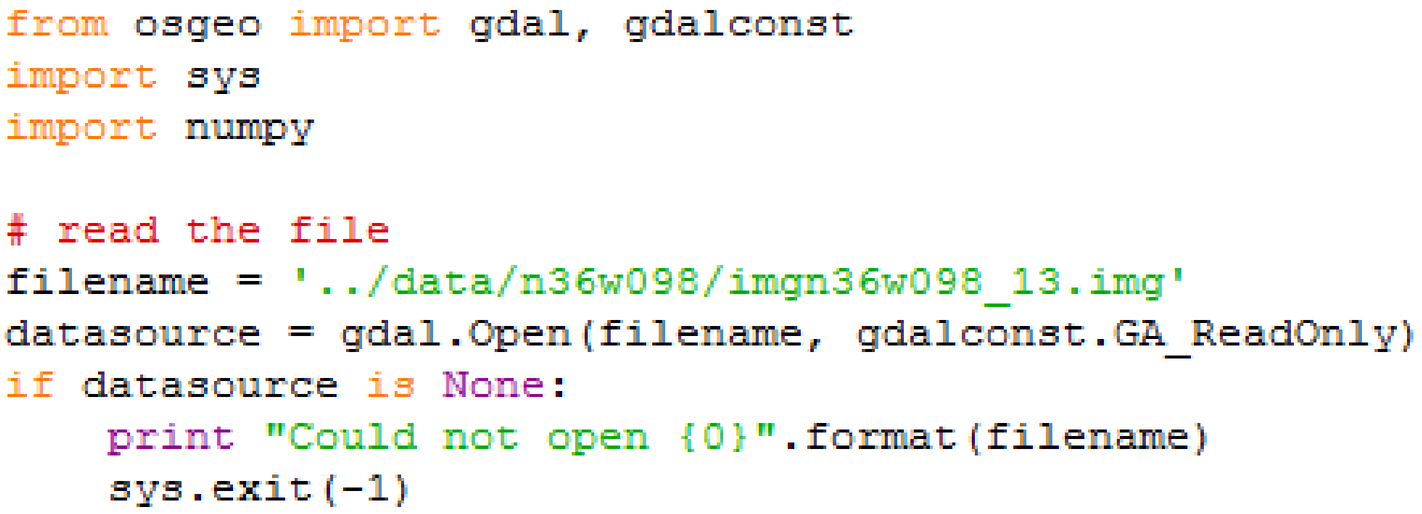
Our goal:

- To crop the raster image
  - Think: create cropped rasters around every lake …



Reading raster

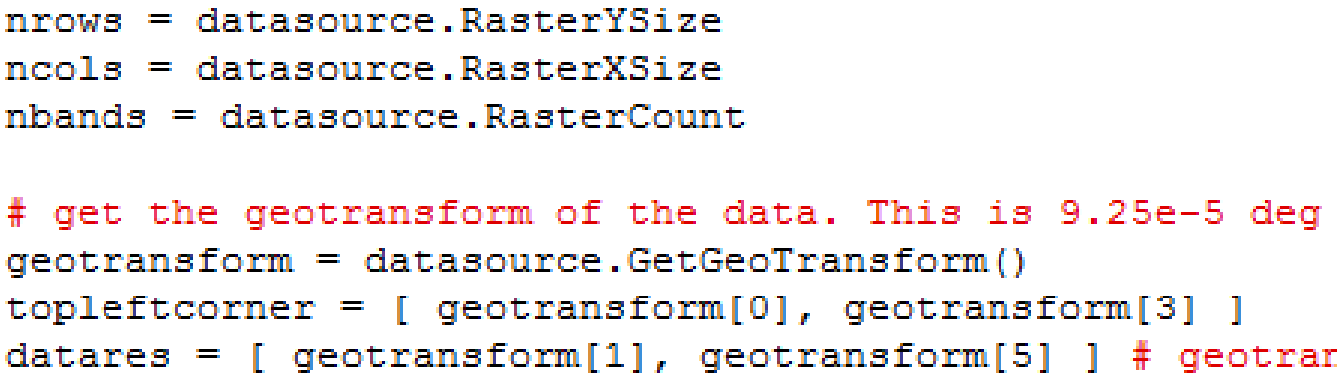
- GDAL is capable of reading and writing many raster formats
  - Here, reading Imagine (.img) files



```python
from osgeo import gdal, gdalconst
import sys
import numpy

# read the file
filename = '../data/n36w098/imgn36w098_13.img'
datasource = gdal.Open(filename, gdalconst.GA_ReadOnly)
if datasource is None:
    print "Could not open {0}".format(filename)
    sys.exit(-1)
```

Basic metadata of raster

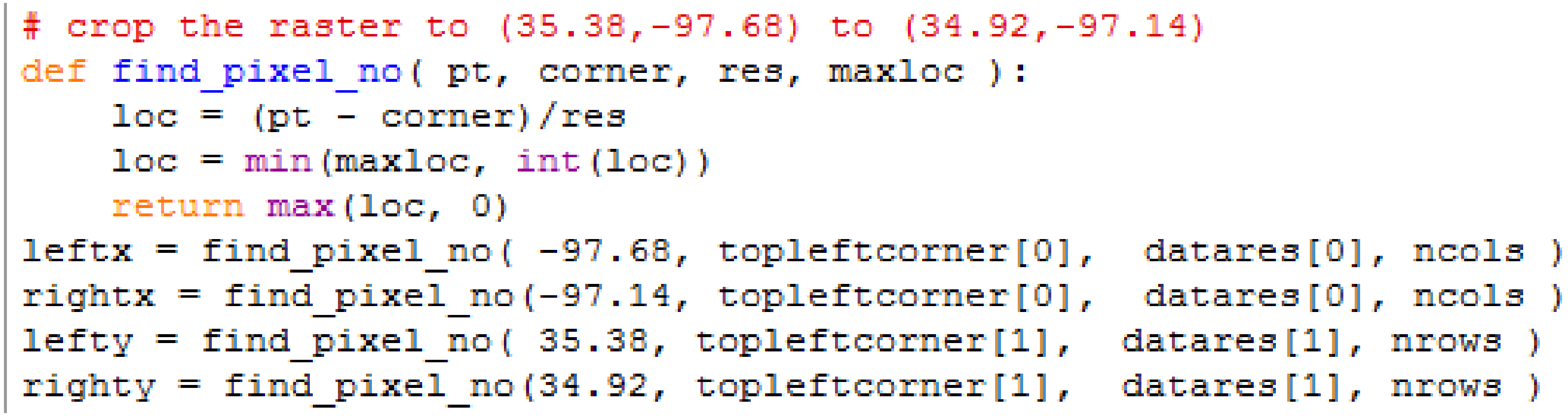
- The 2nd and 4th parameter of geotransform is the rotation
  - Normally zero degrees for true-north pointing data



```
nrows = datasource.RasterYSize
ncols = datasource.RasterXSize
nbands = datasource.RasterCount

# get the geotransform of the data. This is 9.25e-5 deg
geotransform = datasource.GetGeoTransform()
topleftcorner = [ geotransform[0], geotransform[3] ]
datares = [ geotransform[1], geotransform[5] ] # geotrar
```

Cropping raster

- Let’s say that we have found the extent of our region
  - Could look at min, max latitude of the points making up a lake
  - Here, we’ll do the county

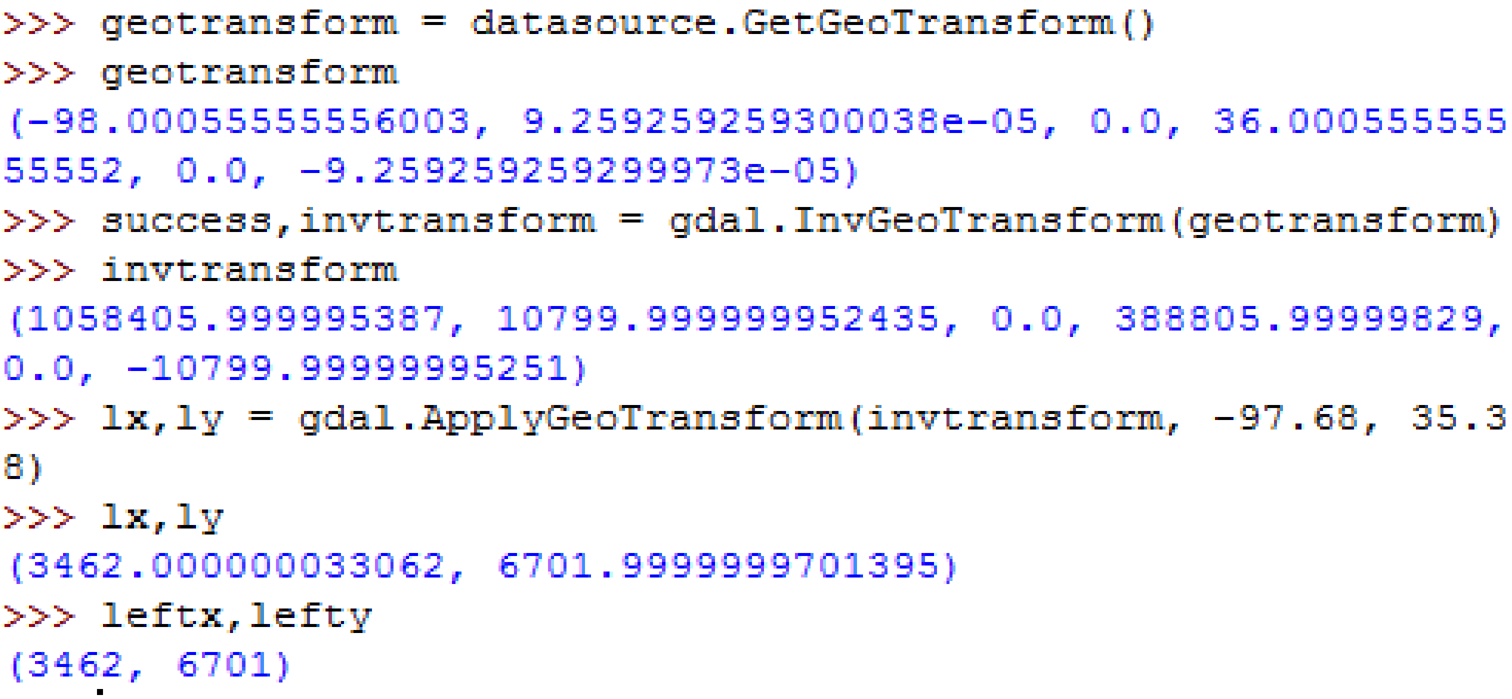


```
# crop the raster to (35.38,-97.68) to (34.92,-97.14)
def find_pixel_no( pt, corner, res, maxloc ):
    loc = (pt - corner)/res
    loc = min(maxloc, int(loc))
    return max(loc, 0)
leftx = find_pixel_no( -97.68, topleftcorner[0],  datares[0], ncols )
rightx = find_pixel_no(-97.14, topleftcorner[0],  datares[0], ncols )
lefty = find_pixel_no( 35.38, topleftcorner[1],  datares[1], nrows )
righty = find_pixel_no(34.92, topleftcorner[1],  datares[1], nrows )
```

- Note that we are careful to not exceed original bounds

Inverse transform

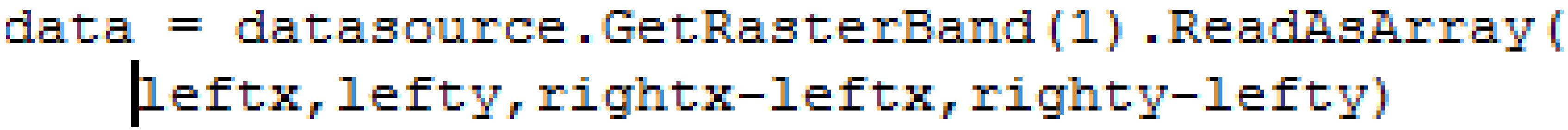
- Could use gdal to do the inversion rather than coding it up



```
>>> geotransform = datasource.GetGeoTransform()
>>> geotransform
(-98.00055555556003, 9.259259259300038e-05, 0.0, 36.000555555
55552, 0.0, -9.259259259299973e-05)
>>> success,invtransform = gdal.InvGeoTransform(geotransform)
>>> invtransform
(1058405.999995387, 10799.999999952435, 0.0, 388805.99999829,
0.0, -10799.99999995251)
>>> lx,ly = gdal.ApplyGeoTransform(invtransform, -97.68, 35.3
8)
>>> lx,ly
(3462.000000033062, 6701.9999999701395)
>>> leftx,lefty
(3462, 6701)
```

Reading in 2D array from raster

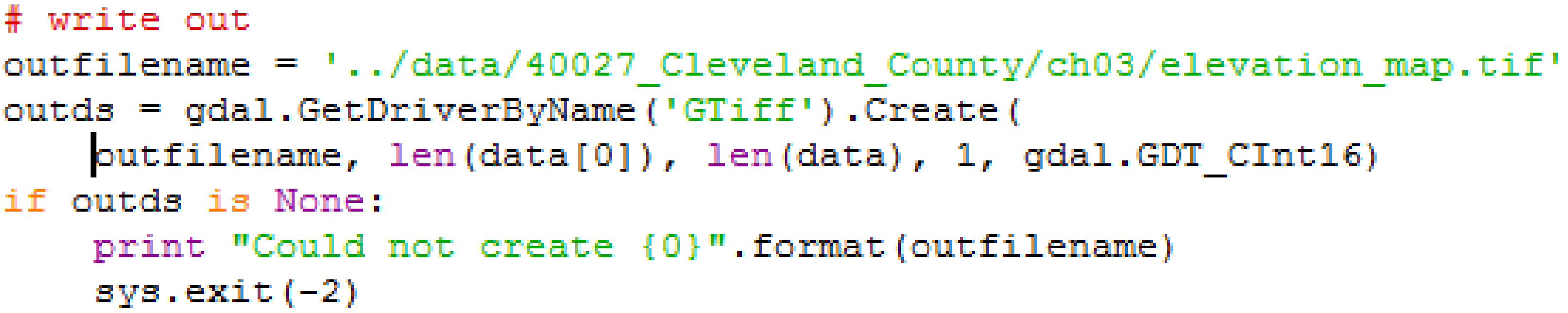
- Specify top-left corner and number of pixels in each direction



```
data = datasource.GetRasterBand(1).ReadAsArray(
    leftx,lefty,rightx-leftx,righty-lefty)
```

Writing out a raster

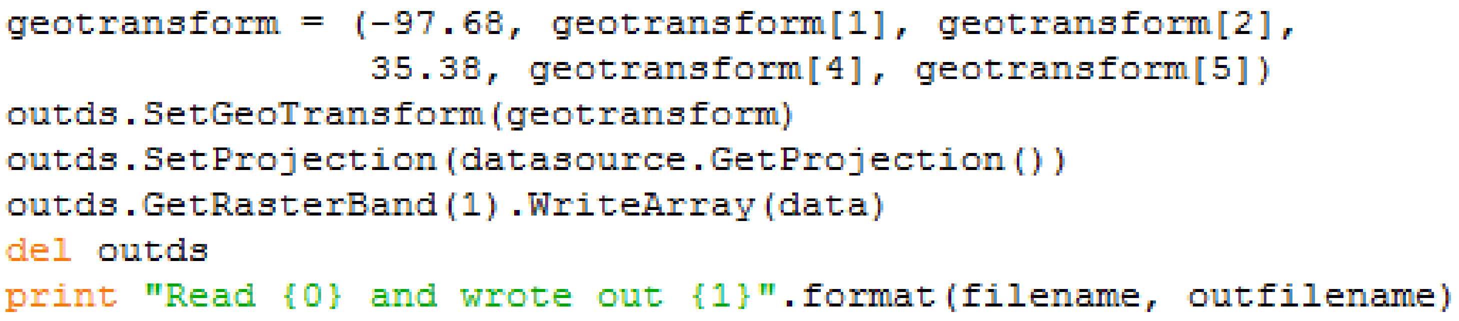
- You will have to choose a format that GDAL is capable of writing out
  - Geotiff a safe choice



```
# write out
outfilename = '../data/40027_Cleveland_County/ch03/elevation_map.tif'
outds = gdal.GetDriverByName('GTiff').Create(
    outfilename, len(data[0]), len(data), 1, gdal.GDT_CInt16)
if outds is None:
    print "Could not create {0}".format(outfilename)
    sys.exit(-2)
```

Specify coordinates & projection

- Make sure to delete the datasource after you are done
  - Cleans up resources, flushes the file



```
geotransform = (-97.68, geotransform[1], geotransform[2],
                35.38, geotransform[4], geotransform[5])
outds.SetGeoTransform(geotransform)
outds.SetProjection(datasource.GetProjection())
outds.GetRasterBand(1).WriteArray(data)
del outds
print "Read {0} and wrote out {1}".format(filename, outfilename)
```

Spatial programming questions

- Given the bare earth elevation data and the depth of a lake at a certain point
  - How would you find the maximum depth in the lake?
  - How about the volume of water in the lake?
    - Does the projection matter?

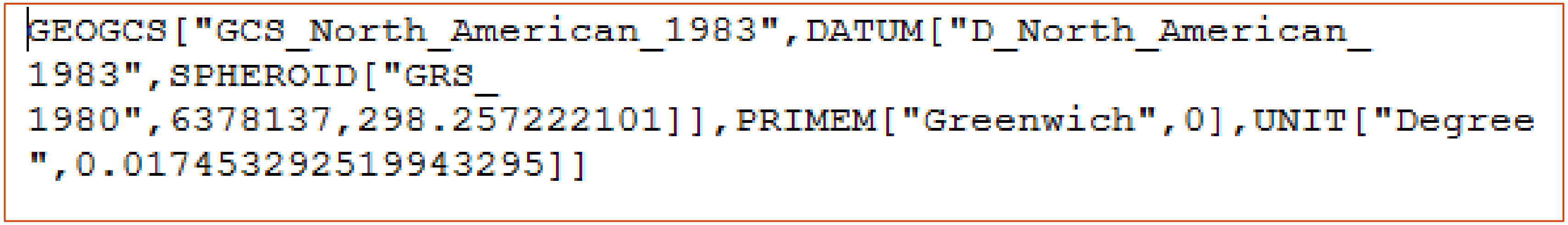
Changing Projections

Different projections

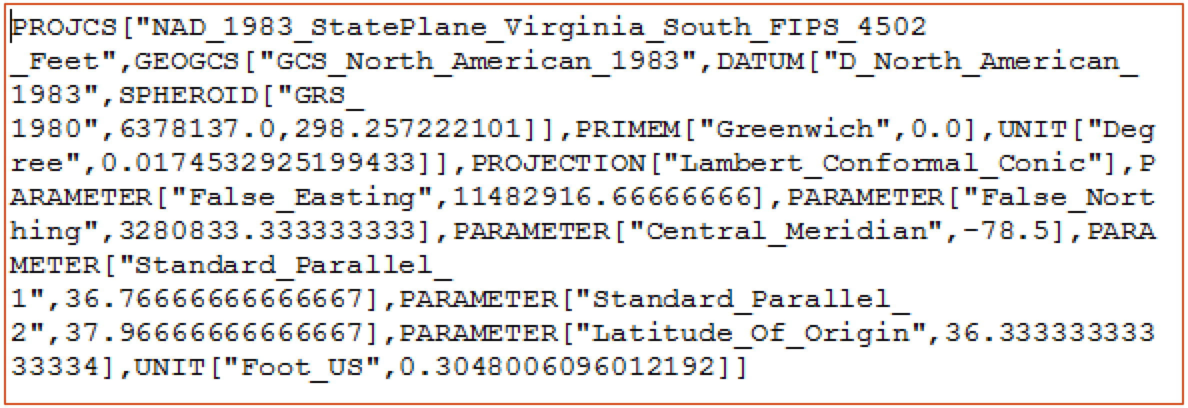
- Often need to deal with datasets in different projections
  - The elevation data is in a well-known geographic coordinate system (WGS-84)
    - And unprojected coordinates (lat-long)
  - The TIGER dataset of water bodies in Cleveland county is in NAD83 spheroid and unprojected coordinates
  - The Isle of Wight fire hydrant locations are also in NAD83 but the projection coordinate system is Lambert Conformal
- What is the difference between a geographic coordinate system (GCS) and a projection coordinate system?
  - Can datasets differ in one or the other? Or both?
  - What does "unprojected coordinates" mean?

Checking the GCS and projection

- You can check the projection of a shapefile by looking at the .prj file associated with it
  - It is in a standard format called WKT ("well known text")



```
GEOGCS["GCS_North_American_1983",DATUM["D_North_American_
1983",SPHEROID["GRS_
1980",6378137,298.257222101]],PRIMEM["Greenwich",0],UNIT["Degree
",0.0174532925199432955]]
```



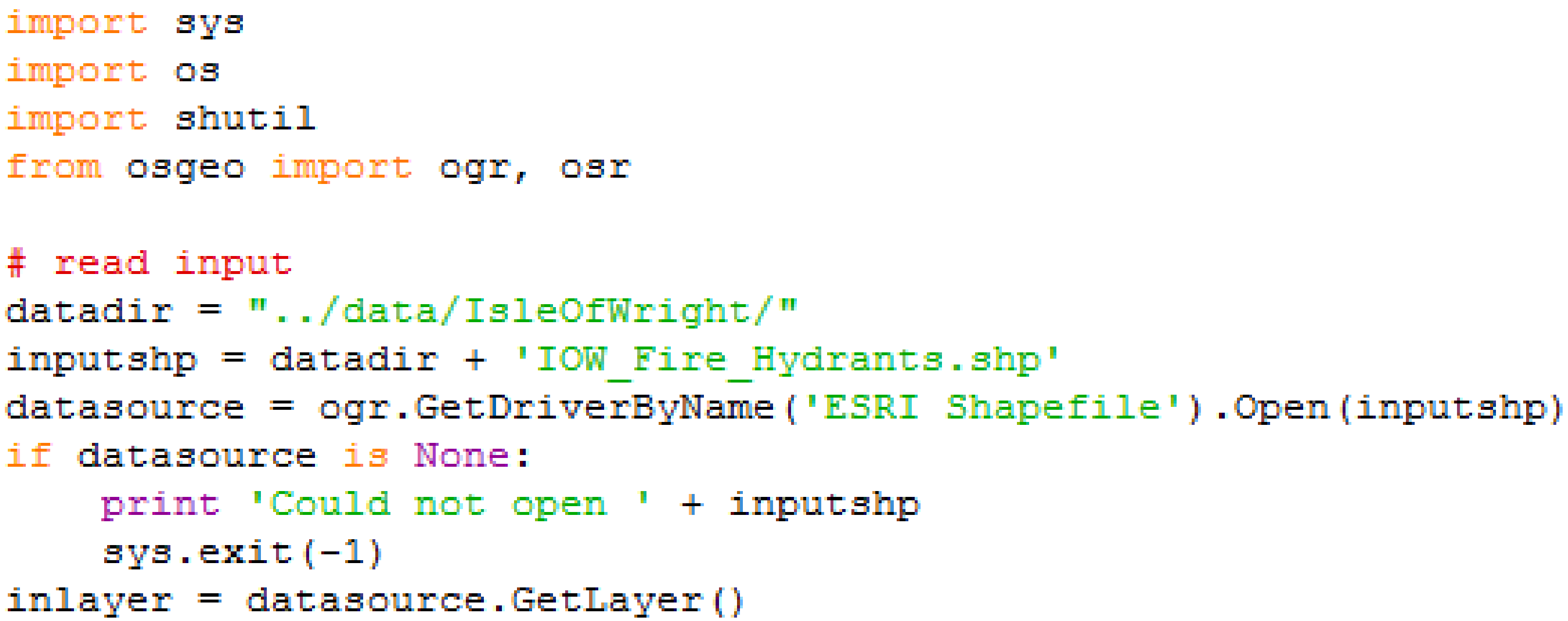
```
PROJCS["NAD_1983_StatePlane_Virginia_South_FIPS_4502
_Feet",GEOGCS["GCS_North_American_1983",DATUM["D_North_American_
1983",SPHEROID["GRS_
1980",6378137.0,298.257222101]],PRIMEM["Greenwich",0.0],UNIT["Deg
ree",0.0174532925199433]],PROJECTION["Lambert_Conformal_Conic"],P
ARAMETER["False_Easting",11482916.66666666],PARAMETER["False_Nort
hing",3280833.333333333],PARAMETER["Central_Meridian",-78.5],PARA
METER["Standard_Parallel_
1",36.76666666666667],PARAMETER["Standard_Parallel_
2",37.96666666666667],PARAMETER["Latitude_Of_Origin",36.333333333
33334],UNIT["Foot_US",0.3048006096012192]]
```

From LCC to Lat-Long

- Recall that I gave you the list of fire hydrant locations in lat-lon for an earlier homework
  - Let's do the conversion that I had to do …

Reading the file

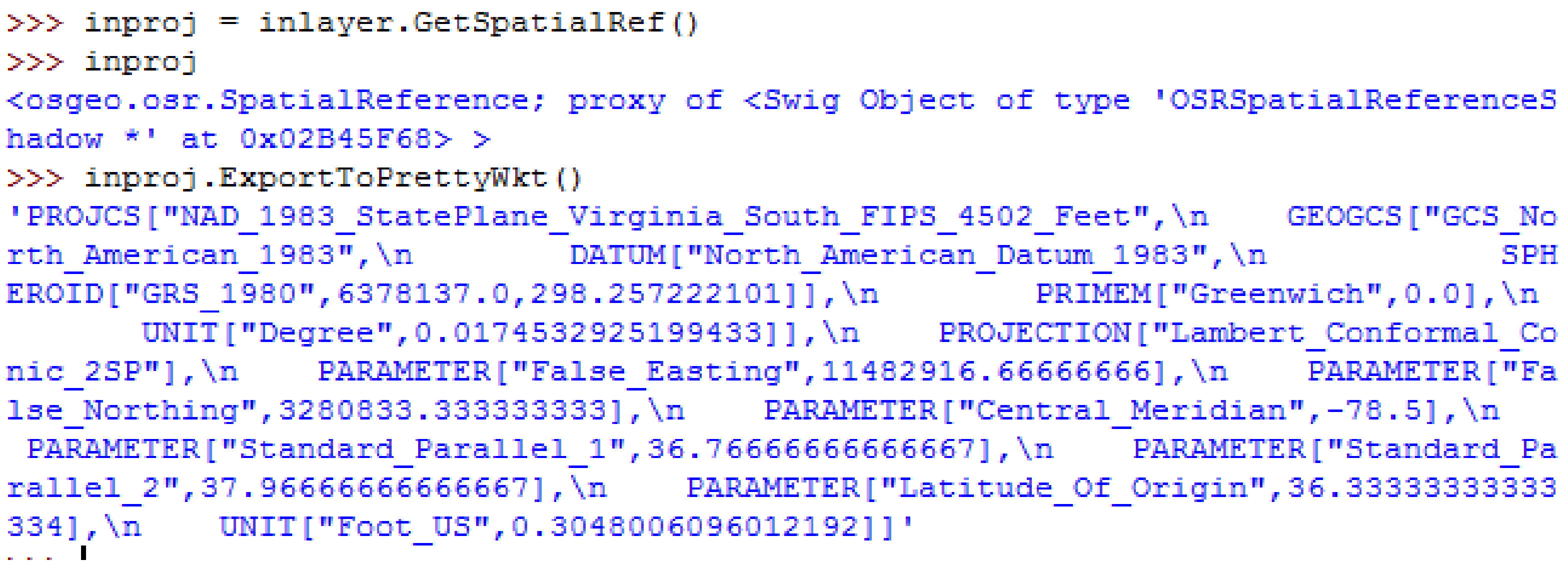
- Use gdal/ogr to read the file
  - OGR is for vector data, OSR for spatial reference (projection)



```
import sys
import os
import shutil
from osgeo import ogr, osr

# read input
datadir = "../data/IsleOfWright/"
inputshp = datadir + 'IOW_Fire_Hydrants.shp'
datasource = ogr.GetDriverByName('ESRI Shapefile').Open(inputshp)
if datasource is None:
    print 'Could not open ' + inputshp
    sys.exit(-1)
inlayer = datasource.GetLayer()
```

Getting the projection

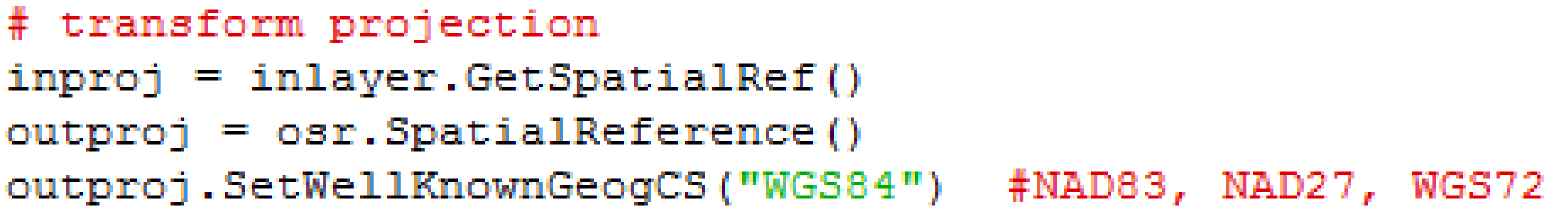
- Ogr calls this the spatial reference
  - And can export to Wkt to see what the .prj would be:



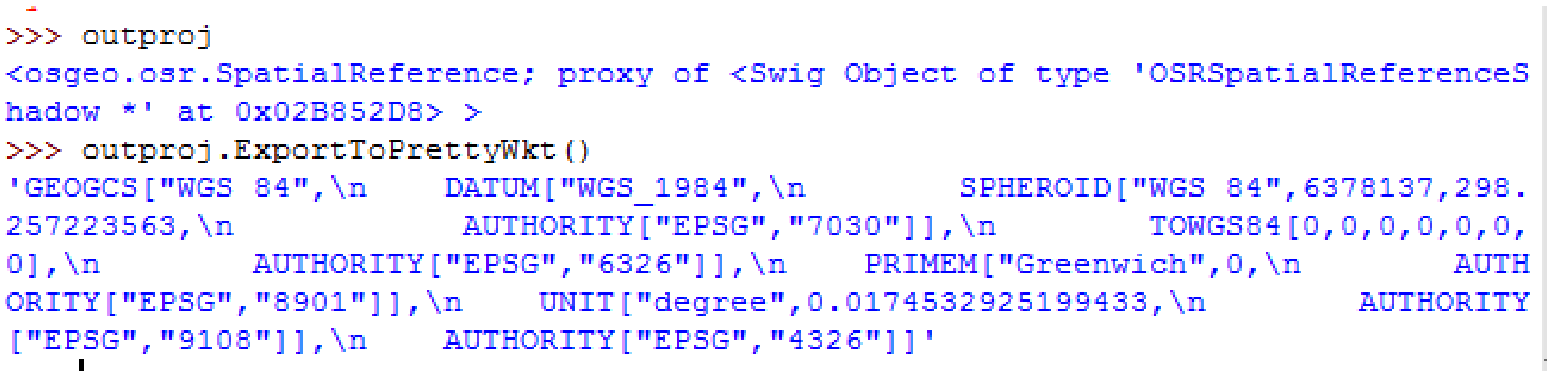
```
>>> inproj = inlayer.GetSpatialRef()
>>> inproj
<osgeo.osr.SpatialReference; proxy of <Swig Object of type 'OSRSpatialReferenceS
hadow *' at 0x02B45F68> >
>>> inproj.ExportToPrettyWkt()
'PROJCS["NAD_1983_StatePlane_Virginia_South_FIPS_4502_Feet",\n    GEOGCS["GCS_No
rth_American_1983",\n        DATUM["North_American_Datum_1983",\n            SPH
EROID["GRS_1980",6378137.0,298.257222101]],\n        PRIMEM["Greenwich",0.0],\n
       UNIT["Degree",0.0174532925199433]],\n    PROJECTION["Lambert_Conformal_Co
nic_2SP"],\n    PARAMETER["False_Easting",11482916.66666666],\n    PARAMETER["Fa
lse_Northing",3280833.333333333],\n    PARAMETER["Central_Meridian",-78.5],\n
 PARAMETER["Standard_Parallel_1",36.76666666666667],\n    PARAMETER["Standard_Pa
rallel_2",37.96666666666667],\n    PARAMETER["Latitude_Of_Origin",36.33333333333
334],\n    UNIT["Foot_US",0.3048006096012192]]'
...
```

Setting up our desired projection

- Create a SpatialReference



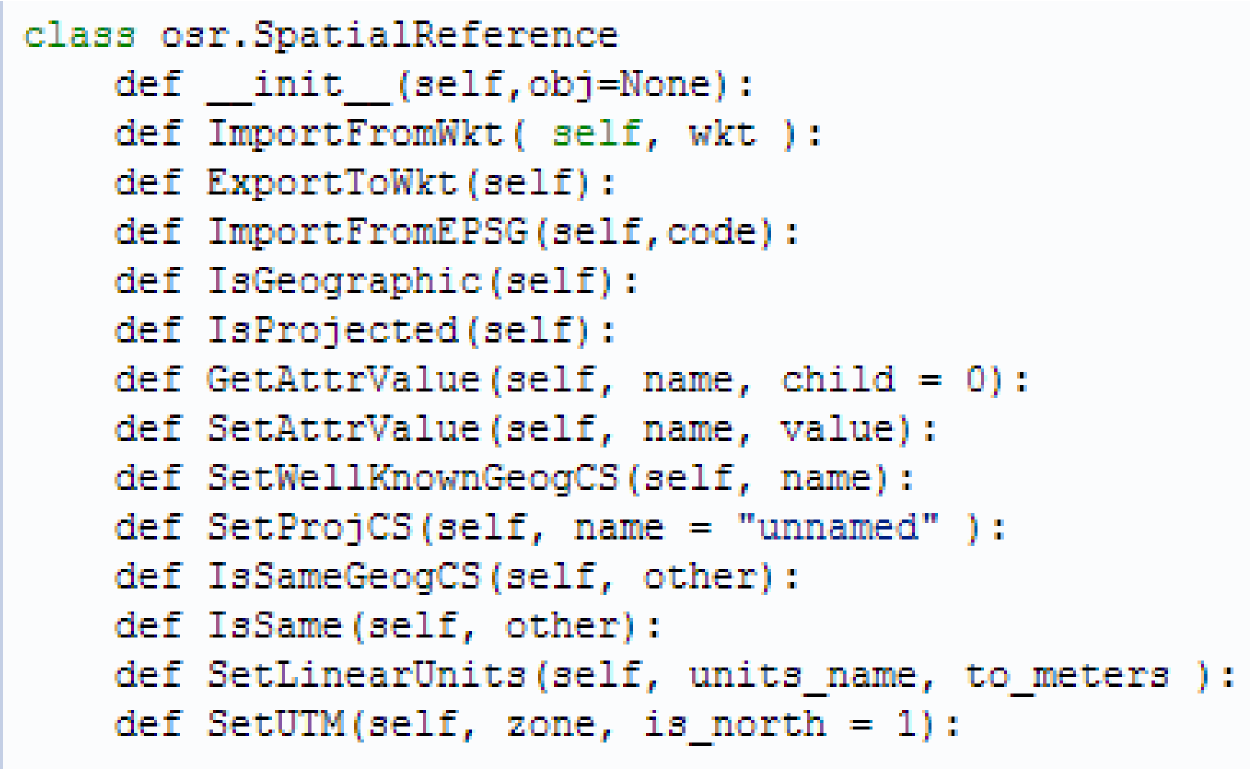
```
# transform projection
inproj = inlayer.GetSpatialRef()
outproj = osr.SpatialReference()
outproj.SetWellKnownGeogCS("WGS84")  #NAD83, NAD27, WGS72
```



```
>>> outproj
<osgeo.osr.SpatialReference; proxy of <Swig Object of type 'OSRSpatialReferenceS
hadow *' at 0x02B852D8> >
>>> outproj.ExportToPrettyWkt()
'GEOGCS["WGS 84",\n    DATUM["WGS_1984",\n        SPHEROID["WGS 84",6378137,298.
257223563,\n            AUTHORITY["EPSG","7030"]],\n        TOWGS84[0,0,0,0,0,0,
0],\n        AUTHORITY["EPSG","6326"]],\n    PRIMEM["Greenwich",0,\n        AUTH
ORITY["EPSG","8901"]],\n    UNIT["degree",0.0174532925199433,\n        AUTHORITY
["EPSG","9108"]],\n    AUTHORITY["EPSG","4326"]]'
```

SpatialReference API

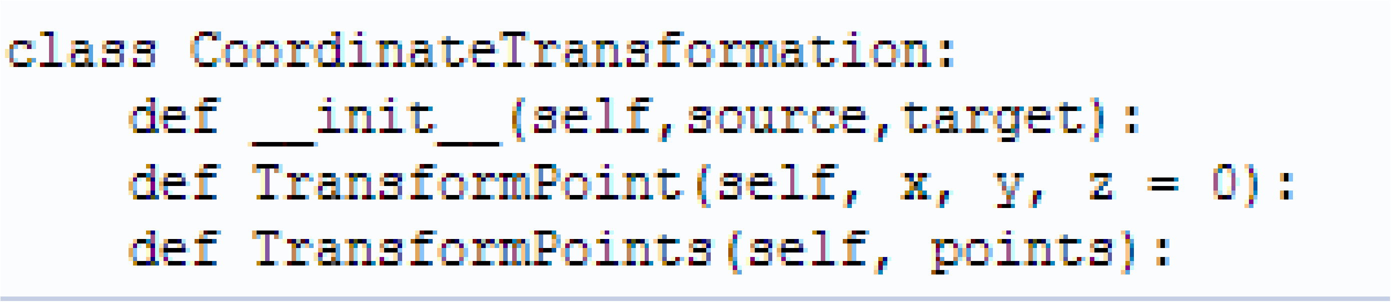
- How would you create a projection for NAD1983 and UTM zone 17?



```
class osr.SpatialReference
    def __init__(self,obj=None):
    def ImportFromWkt( self, wkt ):
    def ExportToWkt(self):
    def ImportFromEPSG(self,code):
    def IsGeographic(self):
    def IsProjected(self):
    def GetAttrValue(self, name, child = 0):
    def SetAttrValue(self, name, value):
    def SetWellKnownGeogCS(self, name):
    def SetProjCS(self, name = "unnamed" ):
    def IsSameGeogCS(self, other):
    def IsSame(self, other):
    def SetLinearUnits(self, units_name, to_meters ):
    def SetUTM(self, zone, is_north = 1):
```

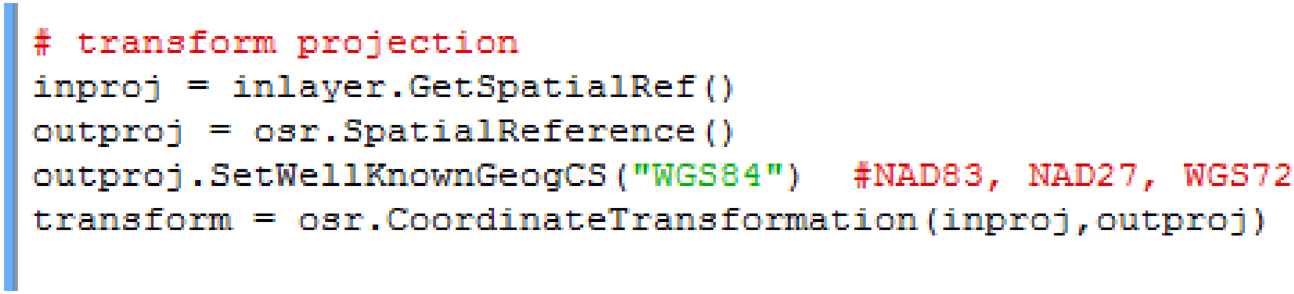
Coordinate Transformation

- To transform between coordinates, use:



```
class CoordinateTransformation:
    def __init__(self,source,target):
    def TransformPoint(self, x, y, z = 0):
    def TransformPoints(self, points):
```

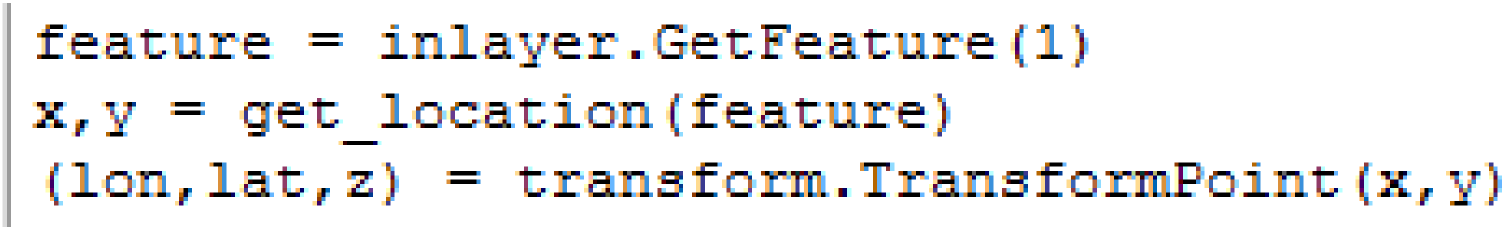
- How would you use this class?



```
# transform projection
inproj = inlayer.GetSpatialRef()
outproj = osr.SpatialReference()
outproj.SetWellKnownGeogCS("WGS84")  #NAD83, NAD27, WGS72
transform = osr.CoordinateTransformation(inproj,outproj)
```

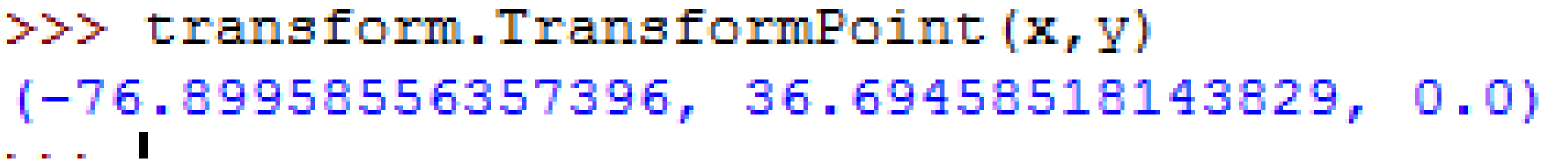
Now transform points one-by-one

- Given the location in the input projection, can get the longitude and latitude in decimal degrees by:



```
feature = inlayer.GetFeature(1)
x,y = get_location(feature)
(lon,lat,z) = transform.TransformPoint(x,y)
```

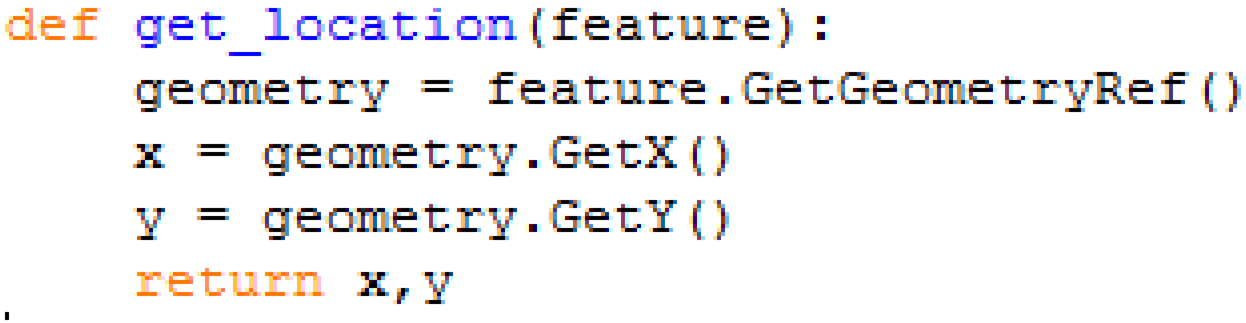
- The output type in Python is not clear from documentation
  - I figured out that it was a List from the interpreter:



```
>>> transform.TransformPoint(x,y)
(-76.89958556357396, 36.69458518143829, 0.0)
... 
```

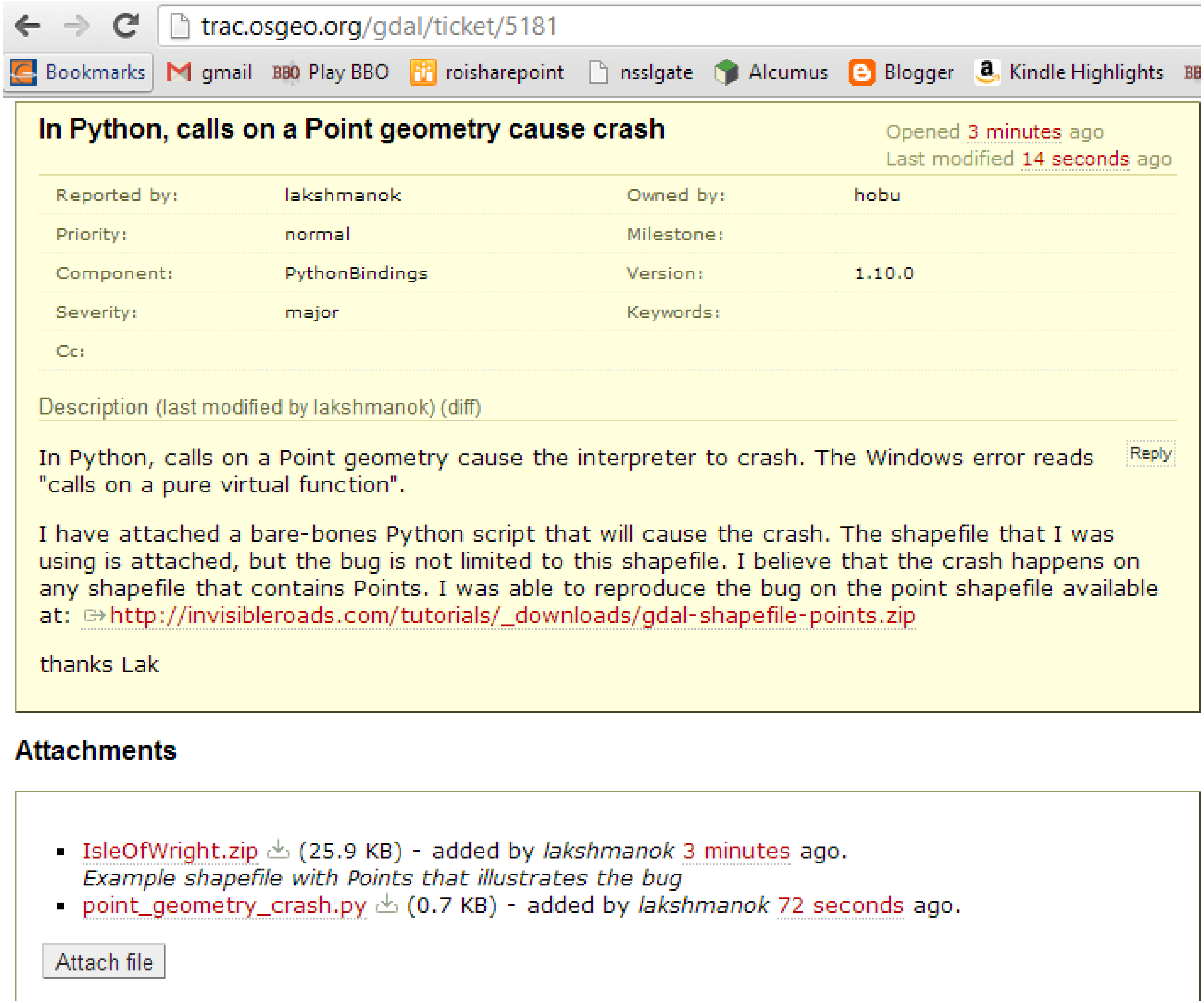
get_location

- To get the location given a feature, this is what “should” work:



```
def get_location(feature):
    geometry = feature.GetGeometryRef()
    x = geometry.GetX()
    y = geometry.GetY()
    return x,y
```

- Unfortunately Python kept crashing on any and all methods on the geometry object returned by GetGeometryRef()
    - What to do?



In Python, calls on a Point geometry cause crash

Opened 3 minutes ago
Last modified 14 seconds ago

Reported by:	lakshmanok	Owned by:	hobu
Priority:	normal	Milestone:	
Component:	PythonBindings	Version:	1.10.0
Severity:	major	Keywords:	
Cc:			

Description (last modified by lakshmanok) (diff)

In Python, calls on a Point geometry cause the interpreter to crash. The Windows error reads "calls on a pure virtual function".

I have attached a bare-bones Python script that will cause the crash. The shapefile that I was using is attached, but the bug is not limited to this shapefile. I believe that the crash happens on any shapefile that contains Points. I was able to reproduce the bug on the point shapefile available at: http://invisibleroads.com/tutorials/_downloads/gdal-shapefile-points.zip

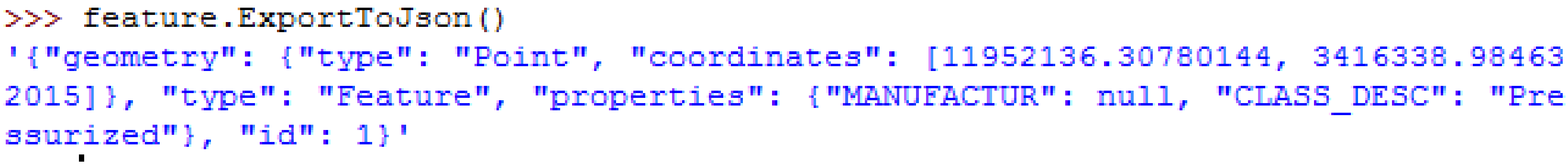
thanks Lak

Attachments

- IsleOfWright.zip (25.9 KB) - added by *lakshmanok* 3 minutes ago.
  Example shapefile with Points that illustrates the bug
- point_geometry_crash.py (0.7 KB) - added by *lakshmanok* 72 seconds ago.

A workaround

- Calls on feature do not crash:

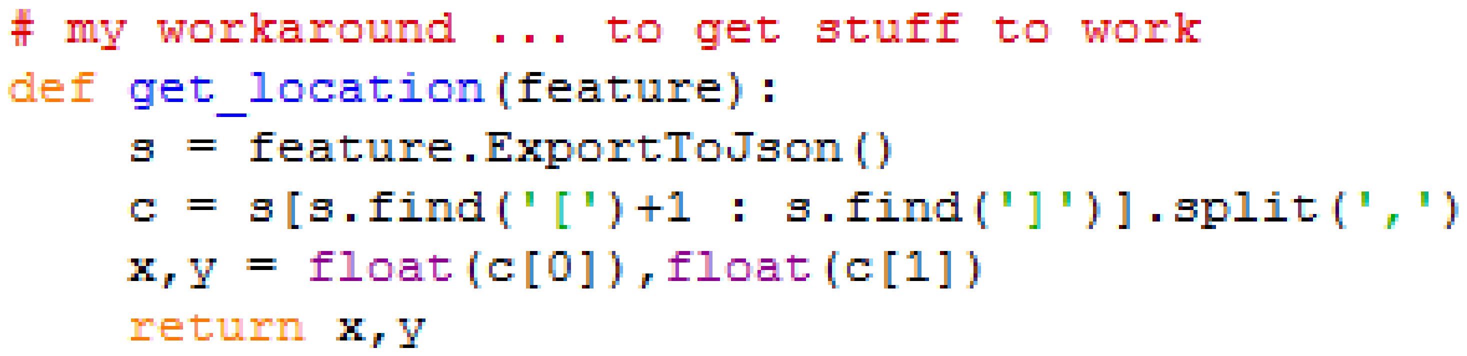


```
>>> feature.ExportToJson()
'{"geometry": {"type": "Point", "coordinates": [11952136.30780144, 3416338.98463
2015]}, "type": "Feature", "properties": {"MANUFACTUR": null, "CLASS_DESC": "Pre
ssurized"}, "id": 1}'
```

- How would you pull out the point locations?

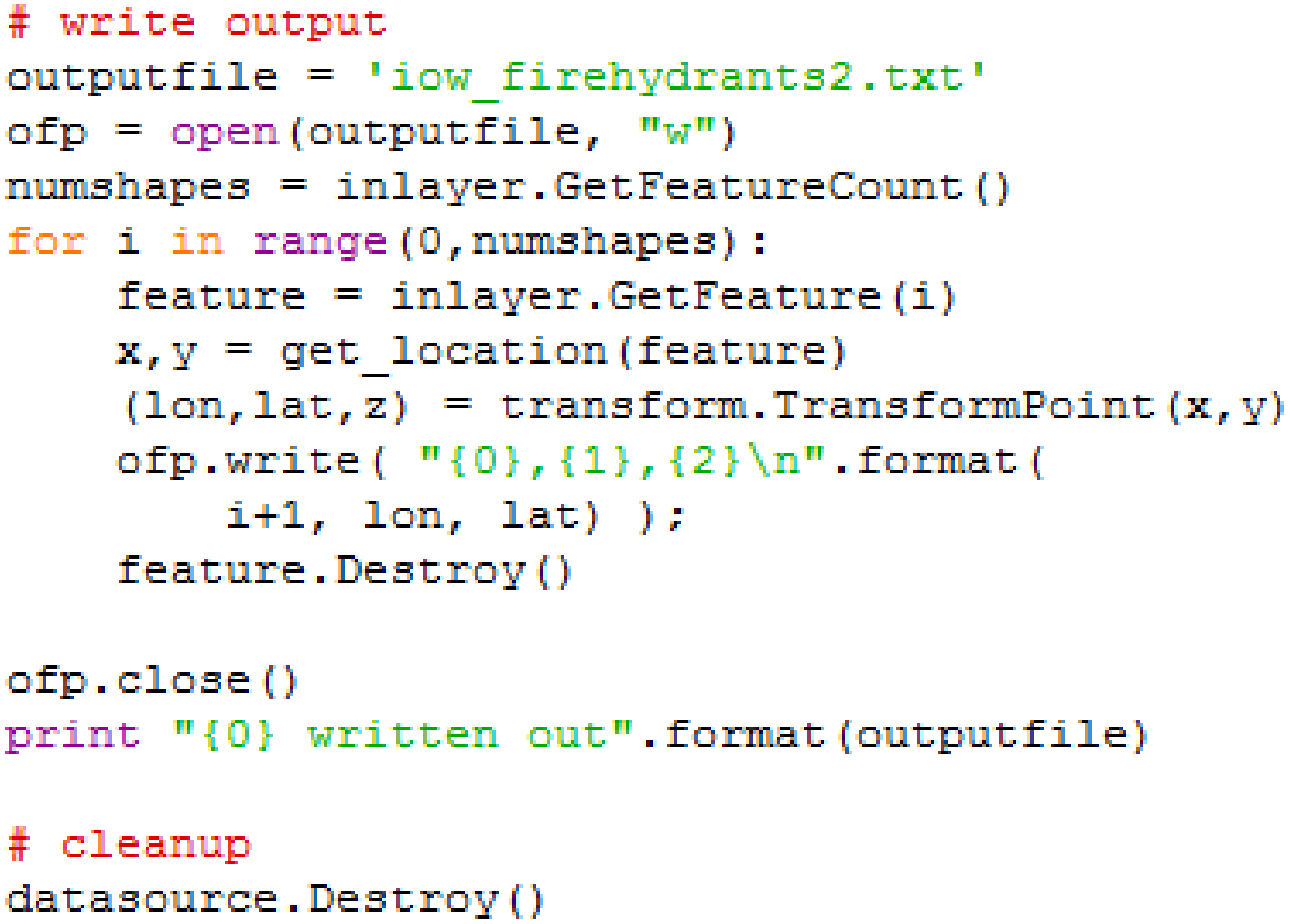
Some string processing

- What is this code doing?



```
# my workaround ... to get stuff to work
def get_location(feature):
    s = feature.ExportToJson()
    c = s[s.find('[')+1 : s.find(']')].split(',')
    x,y = float(c[0]),float(c[1])
    return x,y
```

Writing out a text file



```
# write output
outputfile = 'iow_firehydrants2.txt'
ofp = open(outputfile, "w")
numshapes = inlayer.GetFeatureCount()
for i in range(0,numshapes):
    feature = inlayer.GetFeature(i)
    x,y = get_location(feature)
    (lon,lat,z) = transform.TransformPoint(x,y)
    ofp.write( "{0},{1},{2}\n".format(
        i+1, lon, lat) );
    feature.Destroy()

ofp.close()
print "{0} written out".format(outputfile)

# cleanup
datasource.Destroy()
```

Postscript

- Was not able to reproduce the bug
  - A restart of the Python window solved the problem
  - Workaround no longer needed

Geoprocessing

Common geoprocessing operations

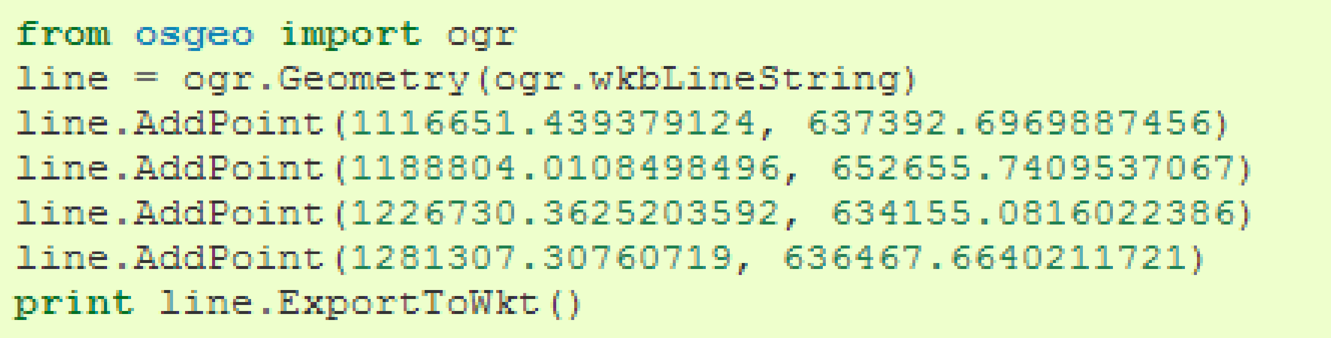
- What do these mean?
  - Buffer
  - Clip
  - Union
  - Intersection
  - Merge
  - Dissolve
- What are these operations on?

Geometry

- The GDAL geometry object is documented here:
  - http://gdal.org/python/osgeo.ogr.Geometry-class.html
- It provides:
  - Create a geometry from a set of points
  - Ways to edit the geometry by adding and removing points
  - Ways to compute length, area
  - Get the boundary as a geometry
  - Find distance to another geometry
  - Find the union, intersection with another geometry
  - Check whether this geometry touches, crosses, is within or overlaps another geometry
  - Buffer a geometry by a distance (in units of shapefile)

Creating geometries

- To create a geometry, specify type of geometry and add points

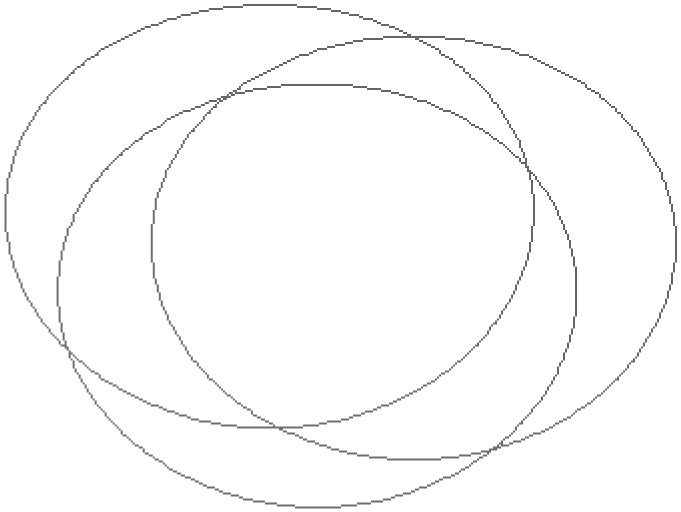


```
from osgeo import ogr
line = ogr.Geometry(ogr.wkbLineString)
line.AddPoint(1116651.439379124, 637392.6969887456)
line.AddPoint(1188804.0108498496, 652655.7409537067)
line.AddPoint(1226730.3625203592, 634155.0816022386)
line.AddPoint(1281307.30760719, 636467.6640211721)
print line.ExportToWkt()
```

http://pcjericks.github.io/py-gdalogr-cookbook/geometry.html

Desired output

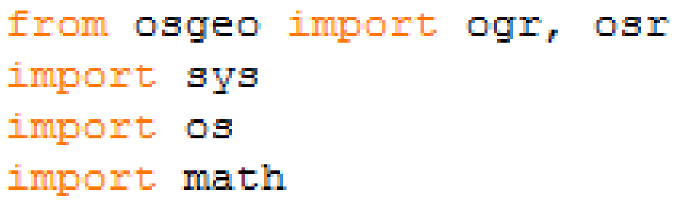
- We want this output
  - Circles of 420km radius centered around each radar in Oklahoma



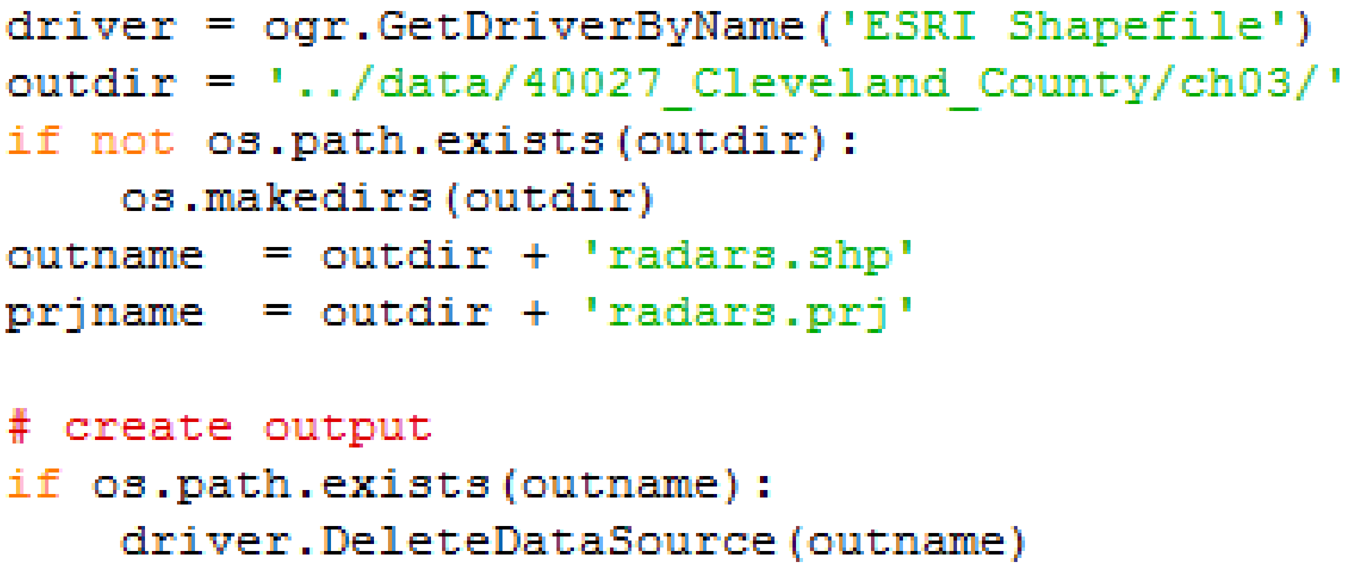
Writing a shapefile from scratch

- To write a shapefile from scratch using OGR:
  - Get driver
  - Create datasource
  - Create layer
  - Create fields (you need at least one field)
  - For each polygon:
    - Make polygon (or whatever geometry)
    - Create a feature and set the fields and geometry on it
    - Provide the feature to the layer
    - Destroy the feature
  - When done, destroy the datasource
  - Also create a .prj file

Preliminaries

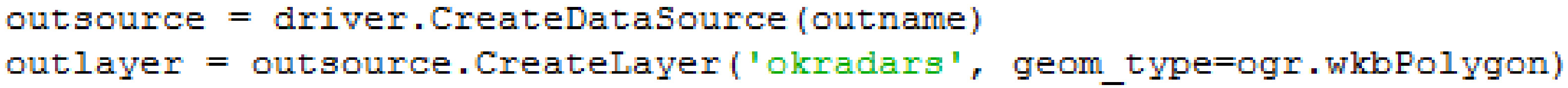


```python
from osgeo import ogr, osr
import sys
import os
import math
```

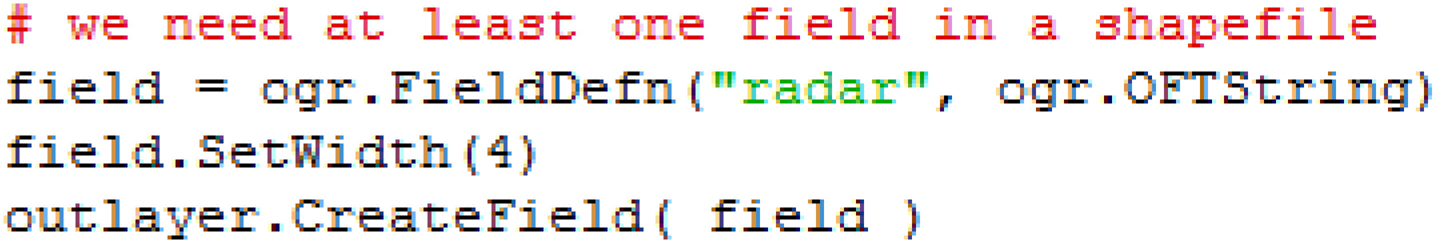


```python
driver = ogr.GetDriverByName('ESRI Shapefile')
outdir = '../data/40027_Cleveland_County/ch03/'
if not os.path.exists(outdir):
    os.makedirs(outdir)
outname  = outdir + 'radars.shp'
prjname  = outdir + 'radars.prj'

# create output
if os.path.exists(outname):
    driver.DeleteDataSource(outname)
```

Datasource, layer, field definition



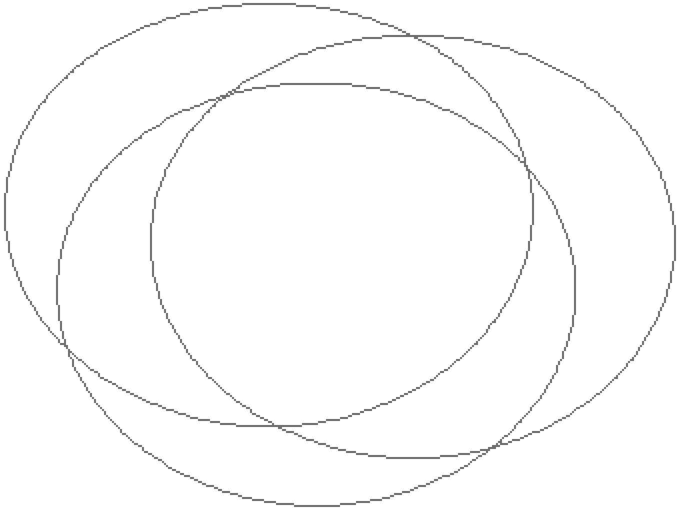
```
outsource = driver.CreateDataSource(outname)
outlayer = outsource.CreateLayer('okradars', geom_type=ogr.wkbPolygon)
```



```
# we need at least one field in a shapefile
field = ogr.FieldDefn("radar", ogr.OFTString)
field.SetWidth(4)
outlayer.CreateField( field )
```

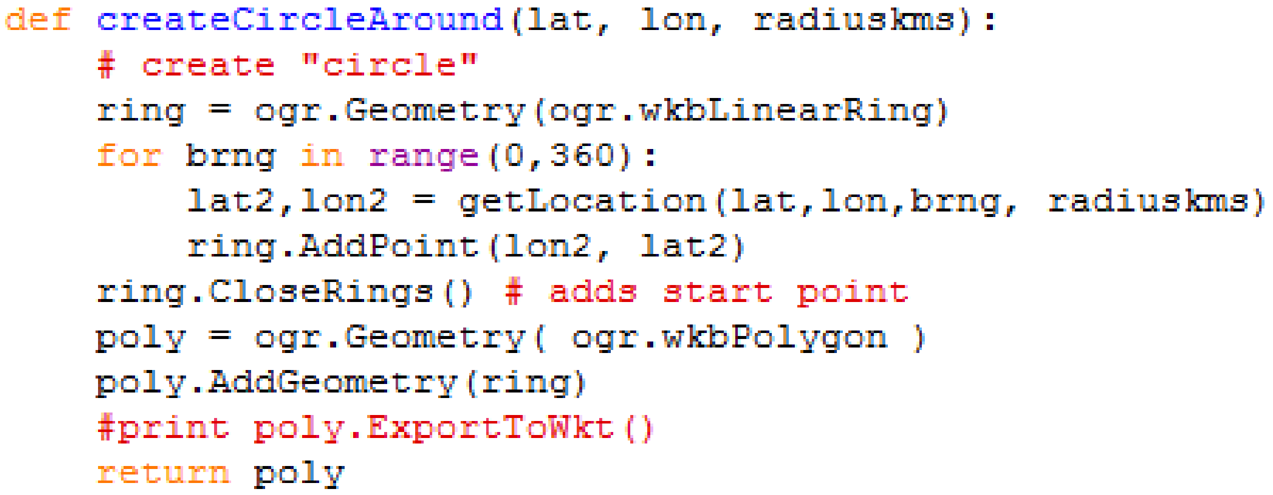
Circles

- Shapefiles do not have circle support
  - Only points, lines and polygons
  - How would you get a circle?



Approximating a circle

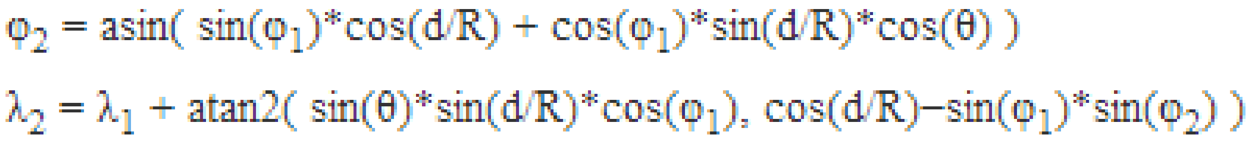
- Can approximate a circle by a polygon with lots of sides
  - A polygon consists of 1 or more linear rings (to account for holes)
  - Each ring needs to be closed



```
def createCircleAround(lat, lon, radiuskms):
    # create "circle"
    ring = ogr.Geometry(ogr.wkbLinearRing)
    for brng in range(0,360):
        lat2,lon2 = getLocation(lat,lon,brng, radiuskms)
        ring.AddPoint(lon2, lat2)
    ring.CloseRings() # adds start point
    poly = ogr.Geometry( ogr.wkbPolygon )
    poly.AddGeometry(ring)
    #print poly.ExportToWkt()
    return poly
```

Location of point given bearing

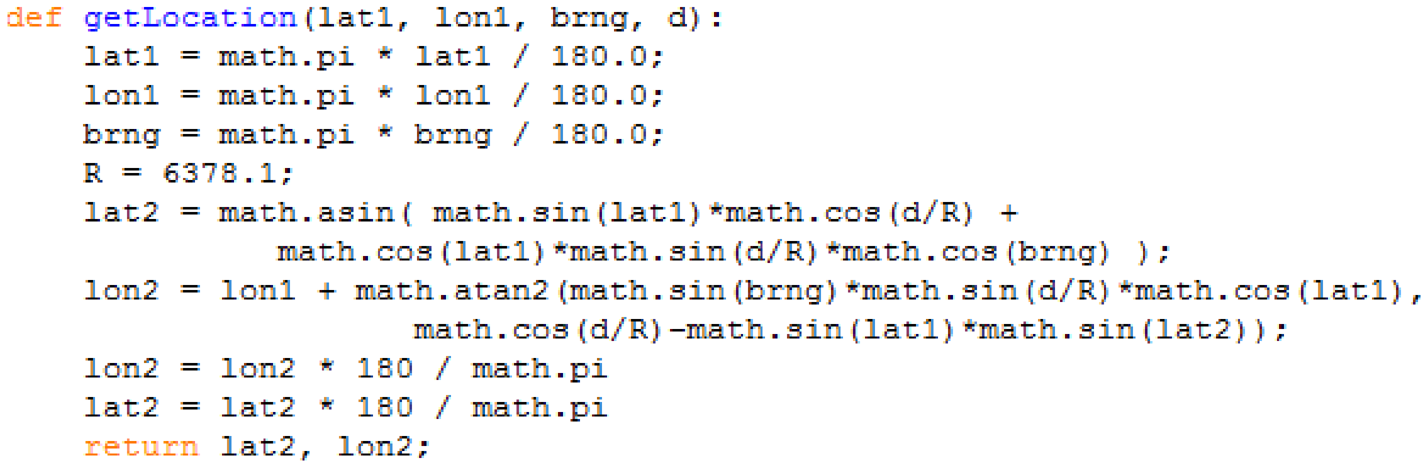
- The location of a point given a bearing and distance is:



$$\varphi_2 = \text{asin}(\ \sin(\varphi_1)^*\cos(d/R) + \cos(\varphi_1)^*\sin(d/R)^*\cos(\theta)\ )$$

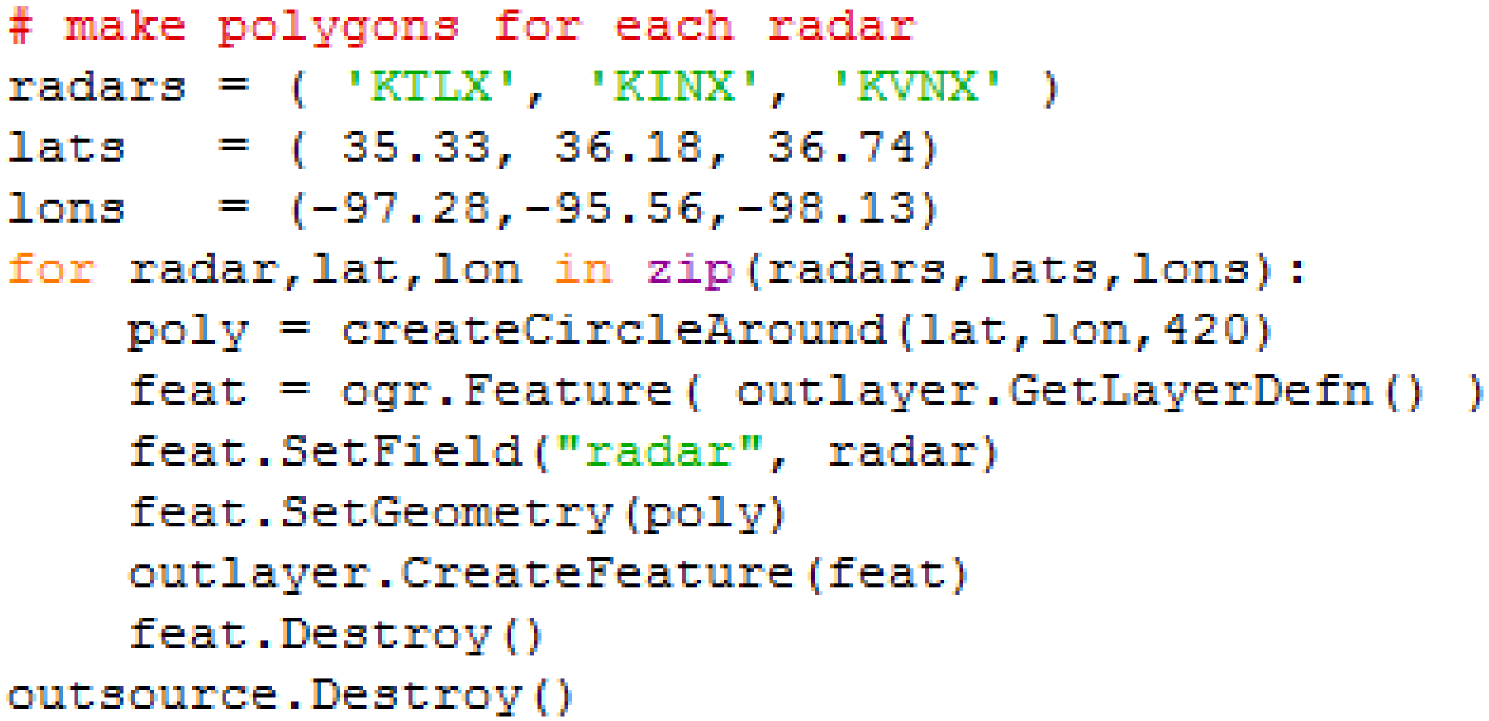
$$\lambda_2 = \lambda_1 + \text{atan2}(\ \sin(\theta)^*\sin(d/R)^*\cos(\varphi_1),\ \cos(d/R)-\sin(\varphi_1)^*\sin(\varphi_2)\ )$$

  - http://www.movable-type.co.uk/scripts/latlong.html
- In Python, remember to convert angles to radians:



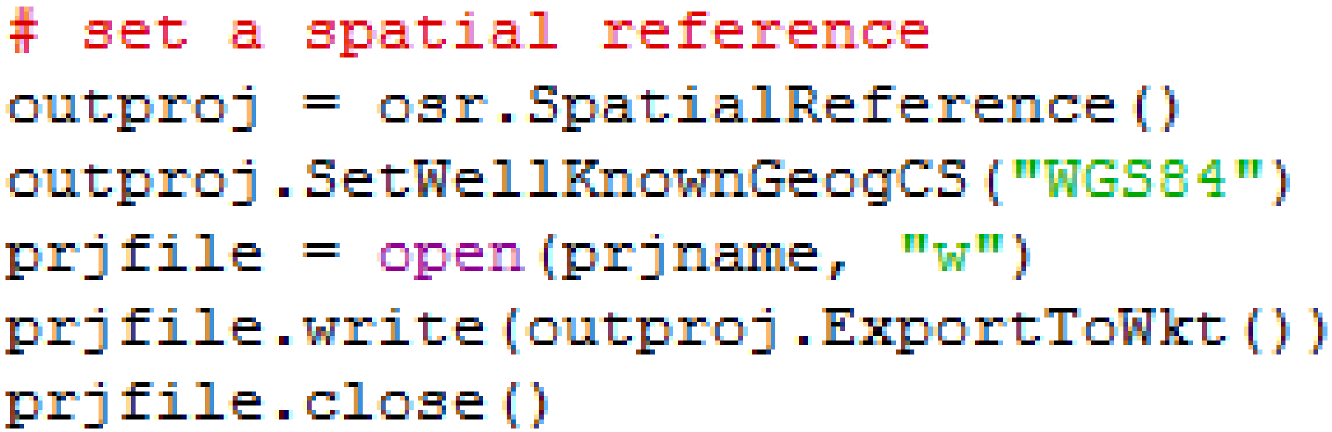
```
def getLocation(lat1, lon1, brng, d):
    lat1 = math.pi * lat1 / 180.0;
    lon1 = math.pi * lon1 / 180.0;
    brng = math.pi * brng / 180.0;
    R = 6378.1;
    lat2 = math.asin( math.sin(lat1)*math.cos(d/R) +
              math.cos(lat1)*math.sin(d/R)*math.cos(brng) );
    lon2 = lon1 + math.atan2(math.sin(brng)*math.sin(d/R)*math.cos(lat1),
                     math.cos(d/R)-math.sin(lat1)*math.sin(lat2));
    lon2 = lon2 * 180 / math.pi
    lat2 = lat2 * 180 / math.pi
    return lat2, lon2;
```

Writing the shapefile



```
# make polygons for each radar
radars = ( 'KTLX', 'KINX', 'KVNX' )
lats   = ( 35.33, 36.18, 36.74)
lons   = (-97.28,-95.56,-98.13)
for radar,lat,lon in zip(radars,lats,lons):
    poly = createCircleAround(lat,lon,420)
    feat = ogr.Feature( outlayer.GetLayerDefn() )
    feat.SetField("radar", radar)
    feat.SetGeometry(poly)
    outlayer.CreateFeature(feat)
    feat.Destroy()
outsource.Destroy()
```

Spatial reference



```
# set a spatial reference
outproj = osr.SpatialReference()
outproj.SetWellKnownGeogCS("WGS84")
prjfile = open(prjname, "w")
prjfile.write(outproj.ExportToWkt())
prjfile.close()
```

Full code

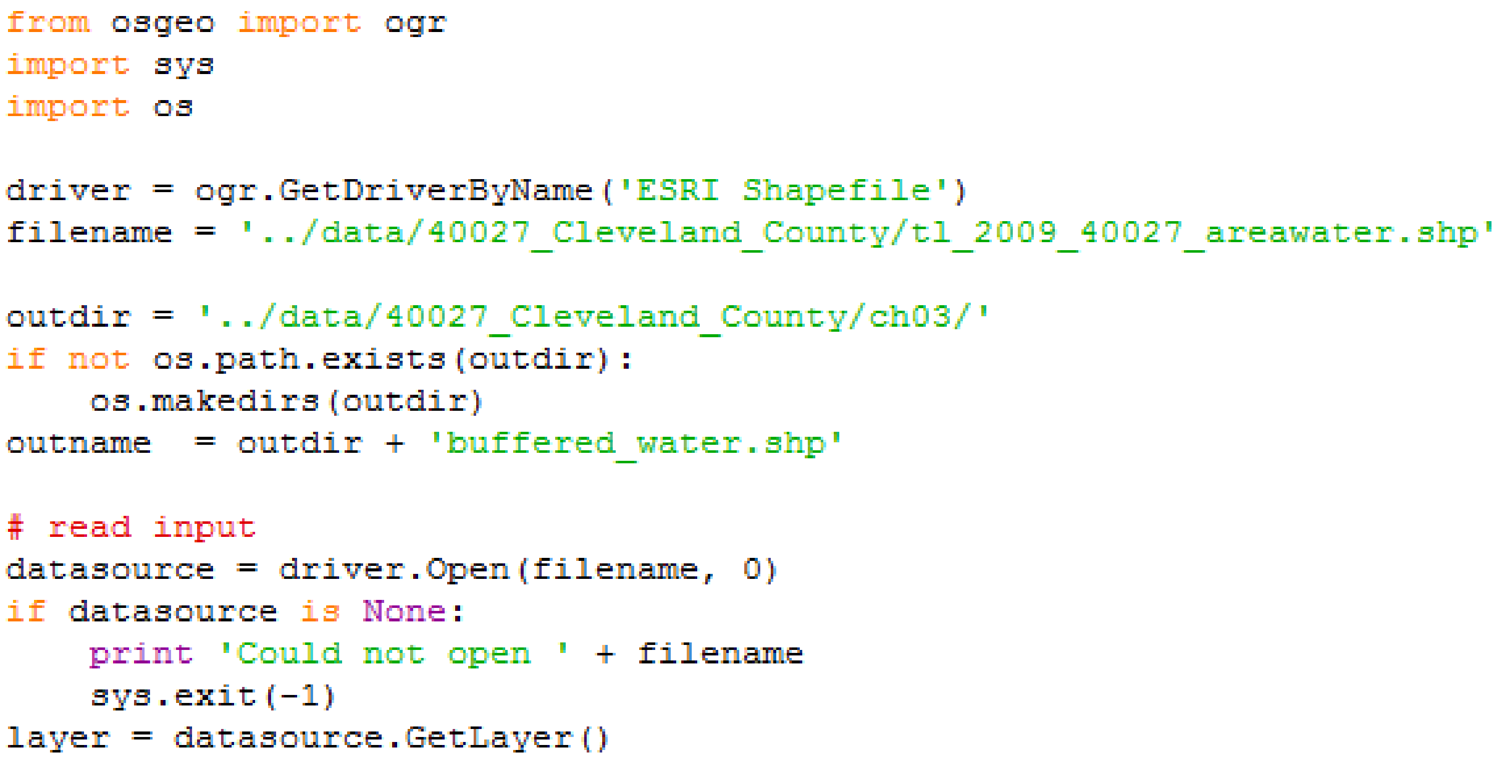
- The full code is at writeradars.py

Buffering

- To buffer a geometry, simply call the Buffer() function
  - Let us buffer all the named lakes
- Steps:
  - Read input file containing all the water bodies
  - Open output file with same geometry type as input
  - Copy the field definitions from the input to the output
  - For each feature in the input that has a name and is not a river
    - Buffer by 0.0005 (the units are units of the input file, so degrees)
    - Write out buffered geometry into output file
  - Clean up

Reading input

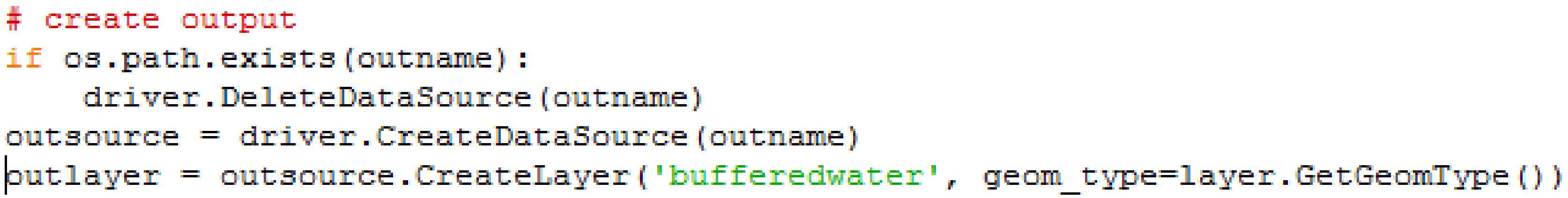
- Should be familiar:



```
from osgeo import ogr
import sys
import os

driver = ogr.GetDriverByName('ESRI Shapefile')
filename = '../data/40027_Cleveland_County/tl_2009_40027_areawater.shp'

outdir = '../data/40027_Cleveland_County/ch03/'
if not os.path.exists(outdir):
    os.makedirs(outdir)
outname  = outdir + 'buffered_water.shp'

# read input
datasource = driver.Open(filename, 0)
if datasource is None:
    print 'Could not open ' + filename
    sys.exit(-1)
layer = datasource.GetLayer()
```

Creating output

- Should also be familiar:

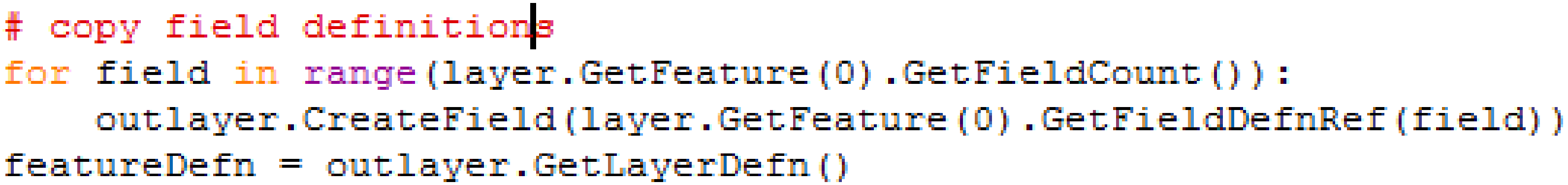


```
# create output
if os.path.exists(outname):
    driver.DeleteDataSource(outname)
outsource = driver.CreateDataSource(outname)
outlayer = outsource.CreateLayer('bufferedwater', geom_type=layer.GetGeomType())
```

- Note how the geometry type is specified

Copy field definitions

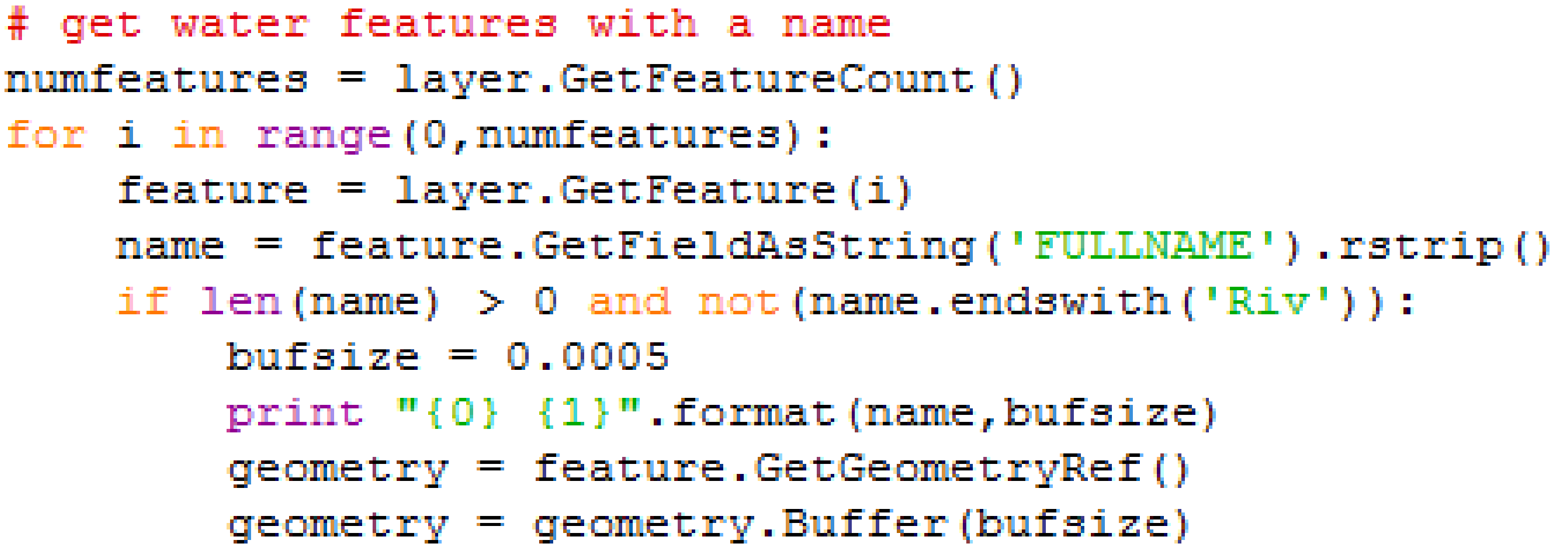
- The output file will have all the attributes of the input



```
# copy field definitions
for field in range(layer.GetFeature(0).GetFieldCount()):
    outlayer.CreateField(layer.GetFeature(0).GetFieldDefnRef(field))
featureDefn = outlayer.GetLayerDefn()
```

Buffer

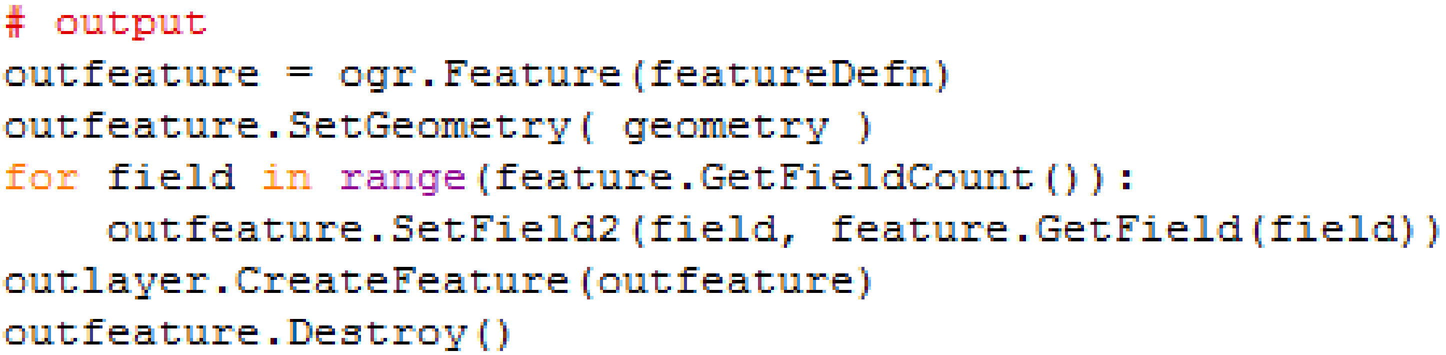
- The 0.0005 is in the original map’s units (decimal degrees)



```
# get water features with a name
numfeatures = layer.GetFeatureCount()
for i in range(0,numfeatures):
    feature = layer.GetFeature(i)
    name = feature.GetFieldAsString('FULLNAME').rstrip()
    if len(name) > 0 and not(name.endswith('Riv')):
        bufsize = 0.0005
        print "{0} {1}".format(name,bufsize)
        geometry = feature.GetGeometryRef()
        geometry = geometry.Buffer(bufsize)
```

- How would you buffer by a specific number of kilometers
    - Say 5 km?

Write out fields and shapes

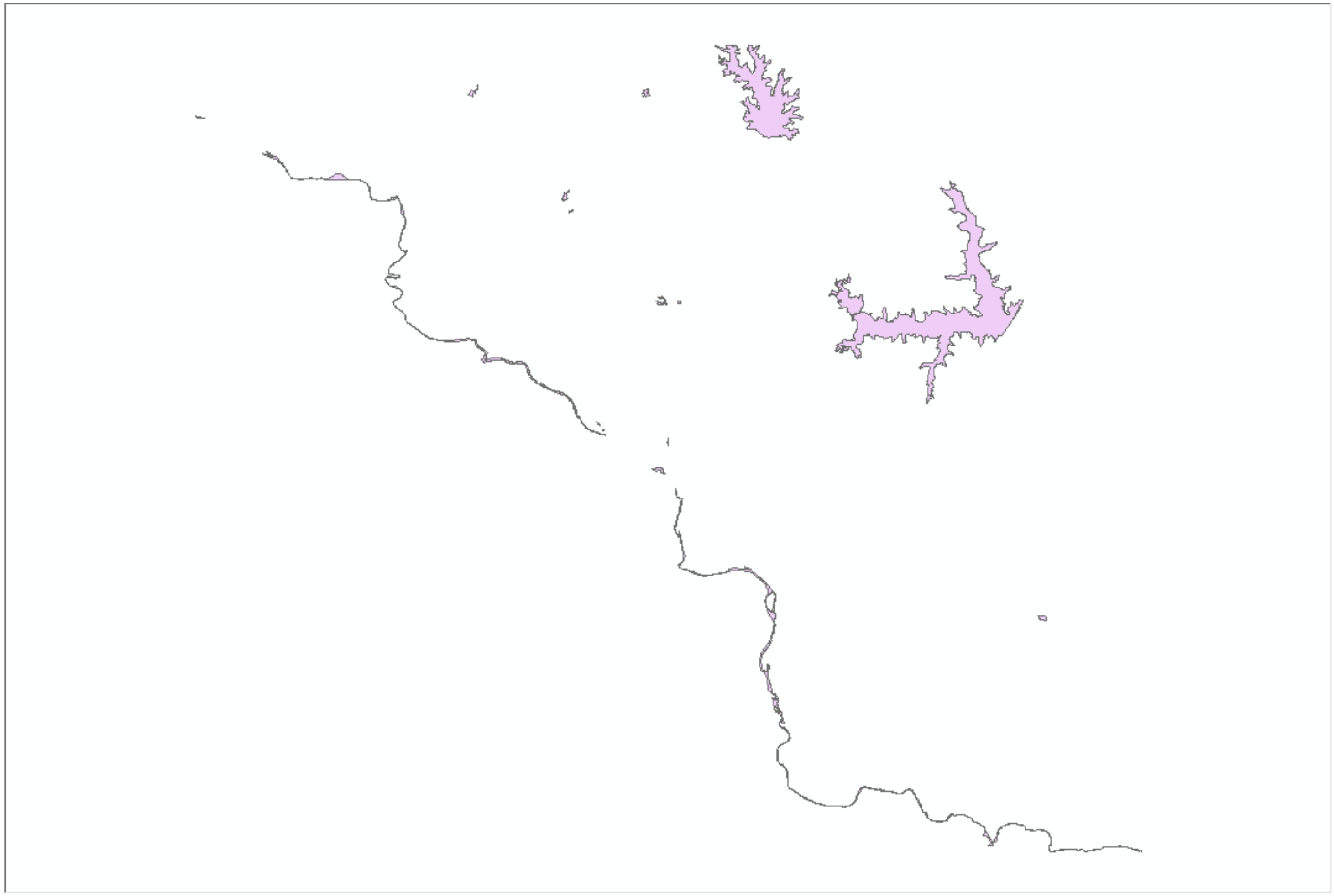


```
# output
outfeature = ogr.Feature(featureDefn)
outfeature.SetGeometry( geometry )
for field in range(feature.GetFieldCount()):
    outfeature.SetField2(field, feature.GetField(field))
outlayer.CreateFeature(outfeature)
outfeature.Destroy()
```

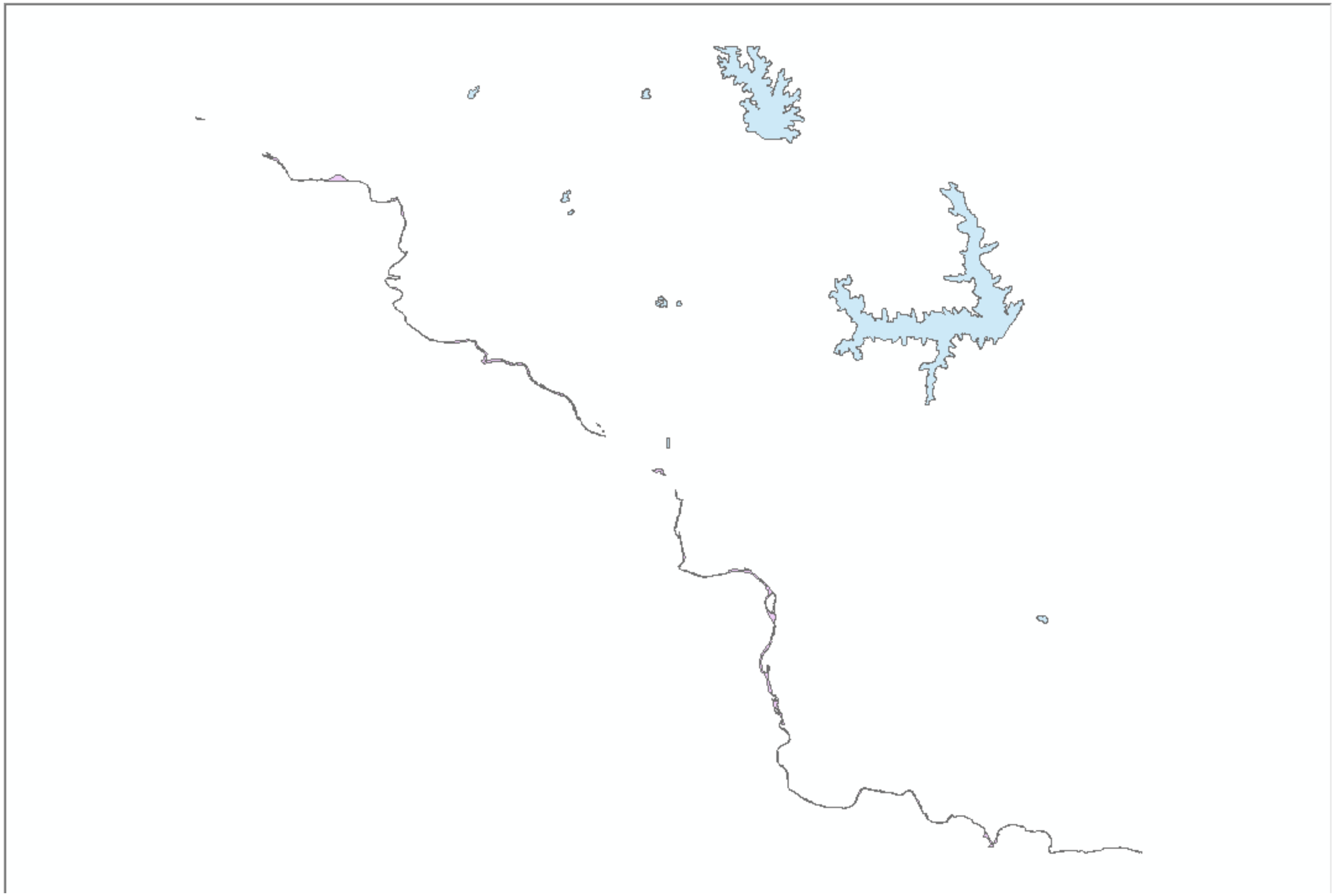
Full code

- The full code is nearwater.py

Before



After

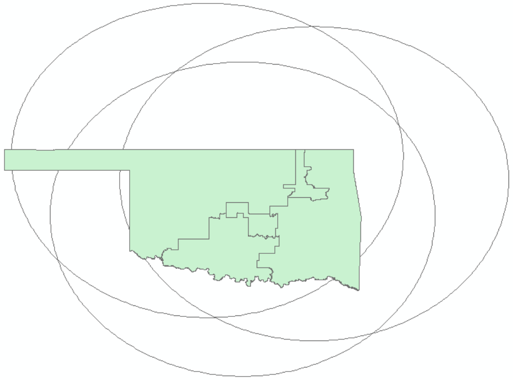


Other geoprocessing

- Look at the API of the Geometry class
  - http://gdal.org/python/osgeo.ogr.Geometry-class.html
- Can you see how to Clip one geometry to the extent of another?
  - How about to the bounding box of another geometry?

Homework

- Create a shapefile that shows range rings out to 420 km but clipped to the state boundary of Oklahoma
  - Obtain NEXRAD locations from http://www.ncdc.noaa.gov/hofnnexrad/HOFNNexradStn
  - You can get an Oklahoma shapefile here
    - http://geo.ou.edu/oeb/Statewide/US_CONG.zip
    - Note that these are congressional districts, not just the state boundary
    - So you will have to find the union of these to form the state geometry



Summary

- We have looked at how to perform basic GIS functions using Open Source Python:
  - shapefile.py for reading and writing shapefiles in pure Python
  - Basemap for plotting data and creating simple maps
  - GDAL/OGR also has advanced GIS functionality
    - A C++ library with Python bindings
    - GDAL for reading and processing raster data
    - OGR for reading and processing vector data
    - Geoprocessing carried out using geometry object
- What are the advantages of open-source packages?
- What are the disadvantages of open-source packages?