

# Python GDAL

**Chris Williams**  
**[chris.williams@Bristol.ac.uk](mailto:chris.williams@Bristol.ac.uk)**

# GDAL

The Geospatial Data Abstraction Library

Cross platform translator library for raster and vector geospatial data

GDAL supports over 50 raster formats

OGR supports over 20 vector formats

Lots of online help

Library access for Python!



# Package installation

Easiest way: use a distribution which contains it ... **Enthought Canopy**  
... so long as you register for an academic license

More information here if you want to install it another way:

<https://pypi.python.org/pypi/GDAL/>

# **osgeo.gdal**

# **Reading raster data in**

Complete Code example: **raster\_io.py**

# osgeo.gdal – reading in raster data

## 1. Import gdal

```
from osgeo import gdal, gdalconst, osr  
from osgeo.gdalconst import *
```

## 2. Set up a driver for reading in a file

- Many raster formats can be imported – see [http://www.gdal.org/formats\\_list.html](http://www.gdal.org/formats_list.html)
- This must be registered before proceeding

```
driver = gdal.GetDriverByName('Gtiff')  
driver.Register()
```

# osgeo.gdal – reading in raster data

3. Open your dataset

```
inDs = gdal.Open(file_name, GA_ReadOnly) <<< Creates an osgeo.gdal Dataset object
```

4. Assign dimensions – useful later on...

```
cols = inDs.RasterXSize  
rows = inDs.RasterYSize  
bands = inDs.RasterCount
```

# osgeo.gdal – reading in raster data

5. Access the Dataset object to get the geotransform info

```
geotransform = inDs.GetGeoTransform()
```

```
geotransform(299685.0, 30.0, 0.0, 8146215.0, 0.0, -30.0)
```

```
# geotransform[0] /* top left x */
```

```
# geotransform[1] /* w-e pixel resolution */
```

```
# geotransform[2] /* rotation, 0 if image is "north up" */
```

```
# geotransform[3] /* top left y */
```

```
# geotransform[4] /* rotation, 0 if image is "north up" */
```

```
# geotransform[5] /* n-s pixel resolution */
```

# osgeo.gdal – reading in raster data

5. Count bands present

```
inDs.RasterCount
```

6. Get band(s)

```
band = inDs.GetRasterBand(1)
```

7. Read in band as an array

```
image_array = band.ReadAsArray(0, 0, cols, rows)
```

**image\_array** is a numpy type array so you can now do what you want with it...



# osgeo.gdal – reading in raster data

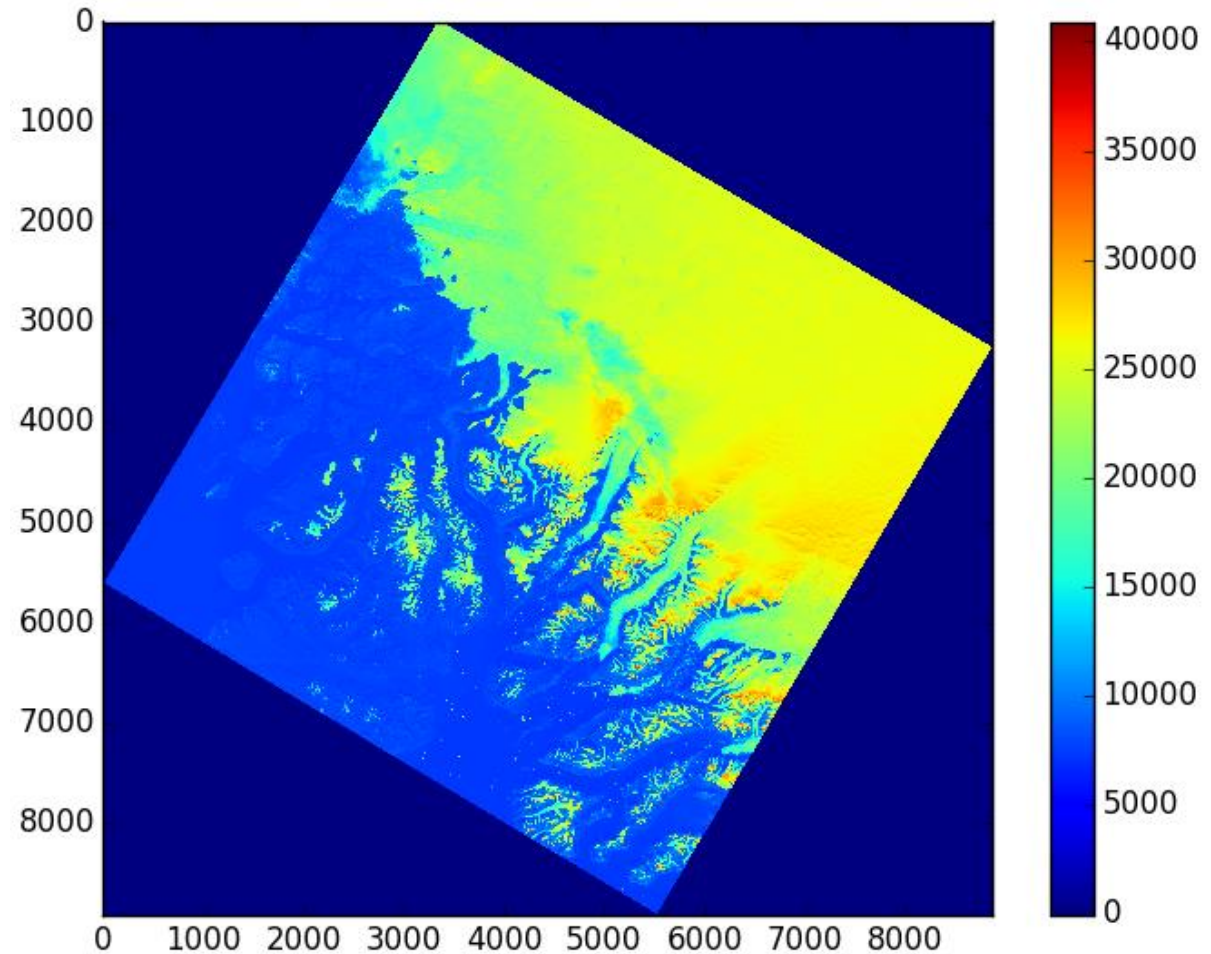
8. Display data using matplotlib.pyplot

```
import matplotlib.pyplot as plt
```

```
plt.imshow(image_array)
```

```
plt.colorbar()
```

```
plt.show()
```



# osgeo.gdal

# Reading raster data out

Complete Code example: [raster\\_io.py](#)

# osgeo.gdal – reading in raster data

So we have now manipulated our values in the array we previously created  
Now we want to write the data out using the **SAME EXTENT**

9. Create a new raster Dataset object

```
file_out='0:/Desktop/python_meeting_group/test_out'  
rows, cols = image_array.shape  
bands = 1
```

<< omit the extension  
otherwise it won't work

```
outDs = driver.Create(file_out, cols, rows, bands, gdal.GDT_Float32)
```

# osgeo.gdal – reading in raster data

10. Set geotransform info (using the same geotransform info as was passed in)

```
outDs.SetGeoTransform(geotransform)
```

11. Set projection info (using the same projection info as was passed in)

```
11.outDs.SetProjection(inDs.GetProjection())
```

12. Get the band from the Dataset object

```
outBand = outDs.GetRasterBand(1)
```

13. Write the data out to create the new raster

```
outBand.WriteArray(image_array)
```

**...you should now have  
your modified raster**

# osgeo.gdal – setting new extent

If you change the extent then you must define a new geotransform...

```
geotransform      = np.zeros(6)
geotransform[0]   = tl_x
geotransform[1]   = post
geotransform[2]   = rotation
geotransform[3]   = tl_y
geotransform[4]   = rotation
geotransform[5]   = -post
```

**You define these variables**

```
geotransform=geotransform.tolist()
```

```
outDs.SetGeoTransform(geotransform)
```

**<< assign the geotransform  
as before**

# osgeo.gdal – setting a new projection

If you change the projection then you must redefine this for the output

Many ways to do this – check out options related to the `osr.SpatialReference` object

```
spatialReference = osr.SpatialReference()
```

```
spatialReference.ImportFromProj4("+proj=stere +lat_0=90 +lat_ts=71 +lon_0=-39  
+k=1 +x_0=0 +y_0=0 +datum=WGS84 +units=m  
+no_defs")
```

```
#spatialReference.SetWellKnownGeogCS( "EPSG:4326" )
```

```
#spatialReference.ImportFromEPSG(4326)
```

```
prj=spatialReference.ExportToWkt()
```

```
outDs.SetProjection(prj)
```

<< Required for serialization and  
transmission of projection definition to other  
packages

[http://www.gdal.org/osr\\_tutorial.html](http://www.gdal.org/osr_tutorial.html)

<http://www.epsg-registry.org/>

<http://geoexamples.blogspot.co.uk/2012/01/creating-files-in-ogr-and-gdal-with.html>

# osgeo.ogr

# Dealing with vector data

Complete Code example: [vector\\_point\\_to\\_poly.py](#)


# osgeo.ogr – reading in some point data

1. Read in points from a space delimited file (using the pandas package)

```
import pandas as pd
```

```
drainage_xy=pd.read_csv('E:/drainage_basins', delim_whitespace=True,  
                        skiprows=7, names=['fid','x','y'])
```

```
fid=drainage_xy['fid'].values  
x=drainage_xy['x'].values  
y=drainage_xy['y'].values
```



create numpy  
arrays of values

Outlines of the full ICESat-based drainage systems generated by the Ice Altimetry group at Goddard Space Flight Center, based on ICESat campaigns L2a-c, L3a-c (2003 Feb - 2005 May).  
Each record: drainage subsystem, lat (deg N), lon (deg E)  
END OF HEADER  
6.1 -533945.261344 -3018815.88986  
6.1 -533943.586014 -3018800.88818  
6.1 -533928.600425 -3018802.48768  
6.1 -533927.050564 -3018787.5806  
6.1 -533897.026835 -3018790.78868  
6.1 -533880.471633 -3018777.36762  
6.1 -533863.863879 -3018763.9558  
6.1 -533815.630871 -3018738.85341  
6.1 -533767.37866 -3018713.63707  
6.1 -533689.28272 -3018691.71132  
6.1 -533596.150608 -3018671.39078  
...etc.

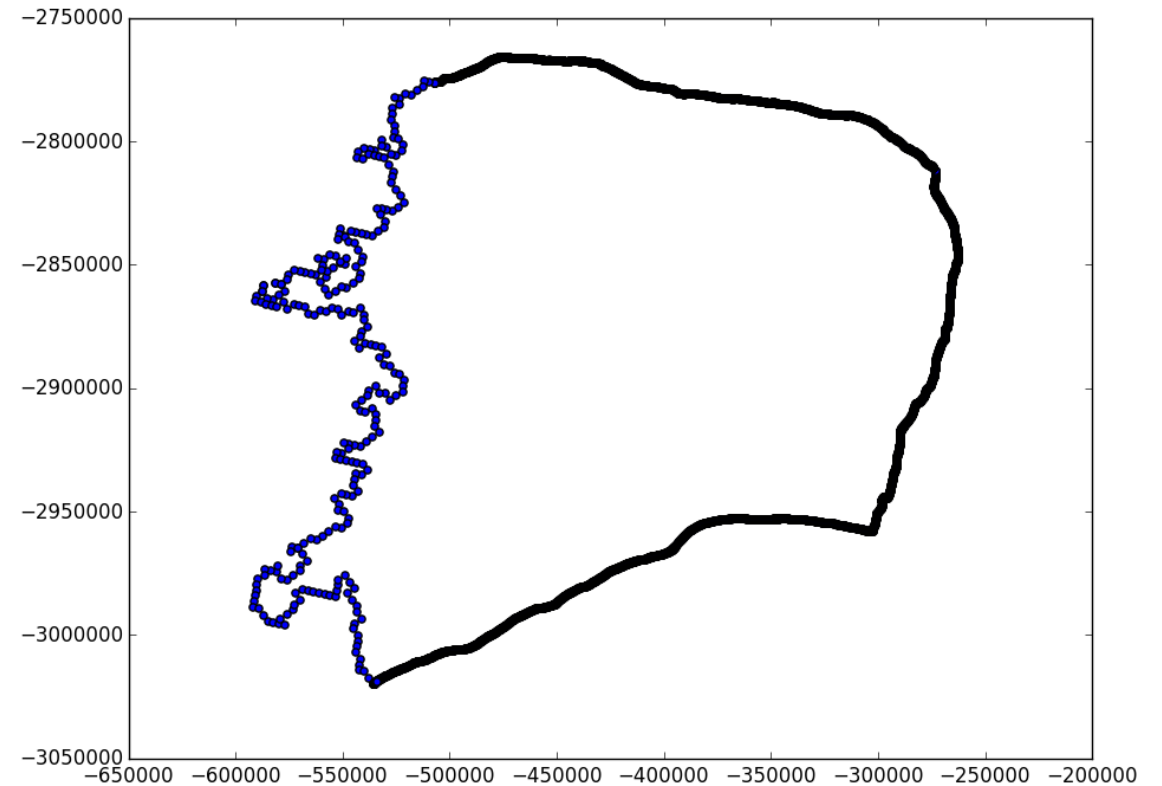


# osgeo.ogr – plot point data

2. Quick plot to see what we have

```
import matplotlib.pyplot as plt
```

```
plt.scatter(lon,lat)  
plt.show()
```





# Structure

**Driver**

**Datasource**

Layer

Feature

Geometry

Polygon/Point

# osgeo.ogr – create layer

4. Create a layer (this will act as a container for any features you create)

```
layer = shapeData.CreateLayer('layer1', spatialReference,  
                              ogr.wkbPolygon)
```



Features inside this will be of type *polygon*



Projection as created before

```
layerDefinition = layer.GetLayerDefn()
```

# Structure

**Driver**

**Datasource**

**Layer**

Feature

Geometry

Polygon/Point

# osgeo.ogr – create ring from points

## 5. Create polygon using points

```
boundary = ogr.Geometry(ogr.wkbLinearRing)
```

<< we use the wkbLinearRing geometry object here

```
#add points to ring
```

```
for i in range(len(fid)):
```

```
    boundary.AddPoint(x_prj_specific[i],y_prj_specific[i])
```

```
boundary.CloseRings()
```

^^ add each point from the x and y vectors to the geometry object

# Structure

**Driver**

**Datasource**

**Layer**

**Feature**

**Geometry**

**Polygon/Point**

# osgeo.ogr – add ring to polygon geometry object

6. Add the ring to a geometry object

```
poly=ogr.Geometry(ogr.wkbPolygon)
```

<< we use the wkbPolygon geometry object here

```
poly.AddGeometry(boundary)
```

<< add the ring we created earlier to the polygon object

```
poly.GetGeometryCount()
```

<< you could add multiple rings e.g. for doughnut shape masks



# Structure

**Driver**

**Datasource**

**Layer**

**Feature**

**Geometry**

**Polygon/Point**

# osgeo.ogr – put feature in layer

7. Put the polygon inside the feature created earlier (point 4)

```
featureIndex = 0
```

```
feature = ogr.Feature(layerDefinition)
```

<< uses the layerDefinition set earlier to hold a polygon

```
feature.SetGeometry(poly)
```

<< poly was our polygon object holding the ring object

```
feature.SetFID(featureIndex)
```

```
layer.CreateFeature(feature)
```

<< put feature inside the layer

# Structure

**Driver**

**Datasource**

**Layer**

**Feature**

**Geometry**

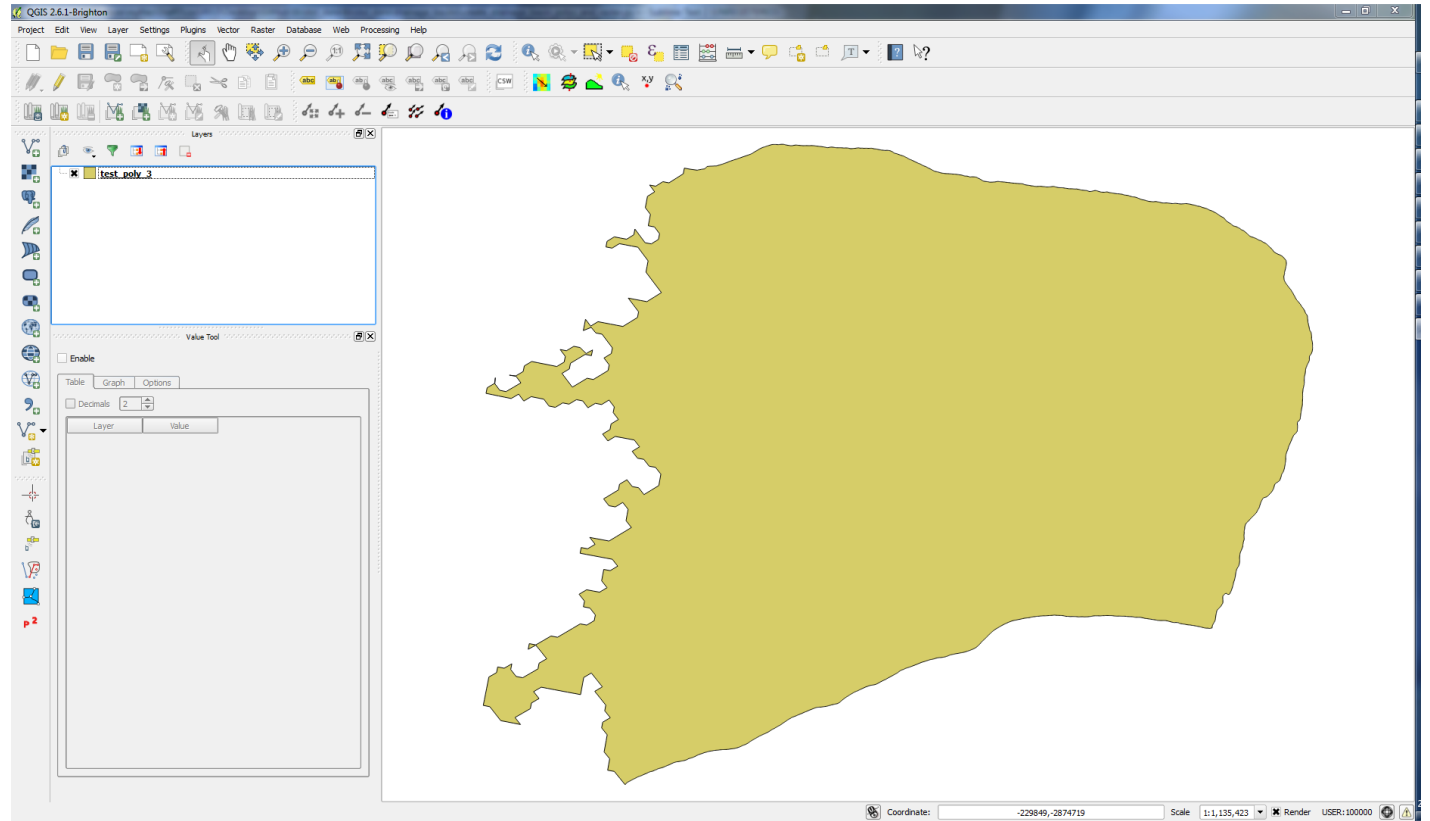
**Polygon/Point**

# osgeo.ogr – close the dataset and view product

8. Flush everything

```
shapeData.Destroy()
```

9. View your polygon!



- Possible to do view in python using the **basemap** package
- Once set up, basemap is great but a bit of a faff to start with

# Documentation and help

Lots of help is available ... this varies in its usefulness!

<http://www.gdal.org/> << **official documentation**

<https://pcjericks.github.io/py-gdalogr-cookbook/> << **GDAL cookbook**

<http://geoinformaticstutorial.blogspot.co.uk/2012/09/reading-raster-data-with-python-and-gdal.html>

<http://www.gis.usu.edu/~chrisg/python/2008/> << very useful but you have to work for some of it!

[http://www2.geog.ucl.ac.uk/~plewis/geogg122/build/html/Chapter4\\_GDAL/OGR\\_Python.html](http://www2.geog.ucl.ac.uk/~plewis/geogg122/build/html/Chapter4_GDAL/OGR_Python.html)

<http://www.epsg-registry.org/> << **EPSG codes**

<http://geoexamples.blogspot.co.uk/2012/01/creating-files-in-ogr-and-gdal-with.html>

# GitHub

**Group Github repository**

**[https://github.com/Chris35Wills/Bristol\\_Geography\\_Python](https://github.com/Chris35Wills/Bristol_Geography_Python)**

**If you have an account, fork and clone this repo**

**raster\_io.py**  
**vector\_point\_to\_poly.py**

**If you have things to contribute then go for it!**

**New to github – check out [this](#)  
Help on contributing to github [here](#)**

