

# Python in GIS

OGR and GDAL



# Learning goals:

After this lesson you should be able to:

- Read and write vector data with OGR
- Perform attribute filters and spatial filters on vector layers
- Convert vector data to other types of formats with OGR
- Create a new raster with GDAL
- Convert to other raster formats with GDAL
- Access a subset of a raster

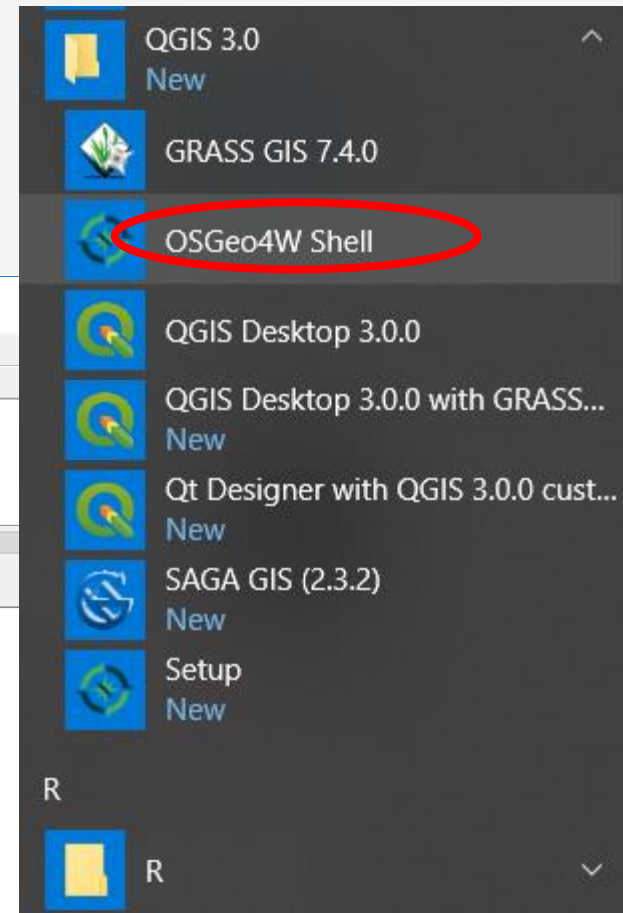
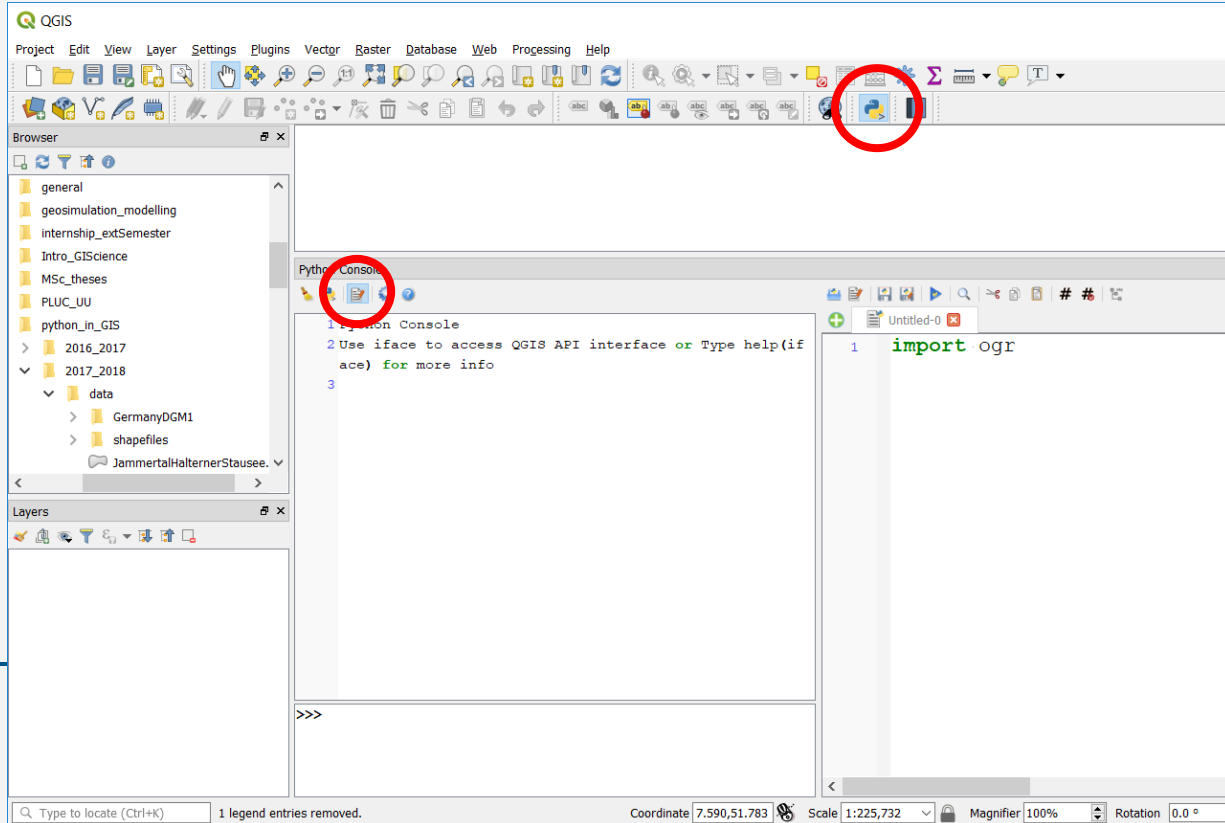
Acknowledgements:

- The book Geoprocessing with Python by Chris Garrard
- The Python GDAL/OGR Cookbook: <https://pcjericks.github.io/py-gdalogr-cookbook/index.html> (Not yet adapted to QGIS 3)

# How to use Python in QGIS?

Two options:

- Via the OSGeo4W Shell in the QGIS menu →
- Within QGIS Desktop



# What are GDAL and OGR?

GDAL:

- means Geospatial Data Abstraction Library
- is a 'translator library' for raster and vector geospatial data
- but is mostly referred to for the raster handling part

OGR:

- used to stand for OpenGIS Simple Features Reference Implementation
- is a part of the GDAL library
- provides access to a large number of vector file formats

The Python API is not extremely well documented, but parallels the C/C++ APIs

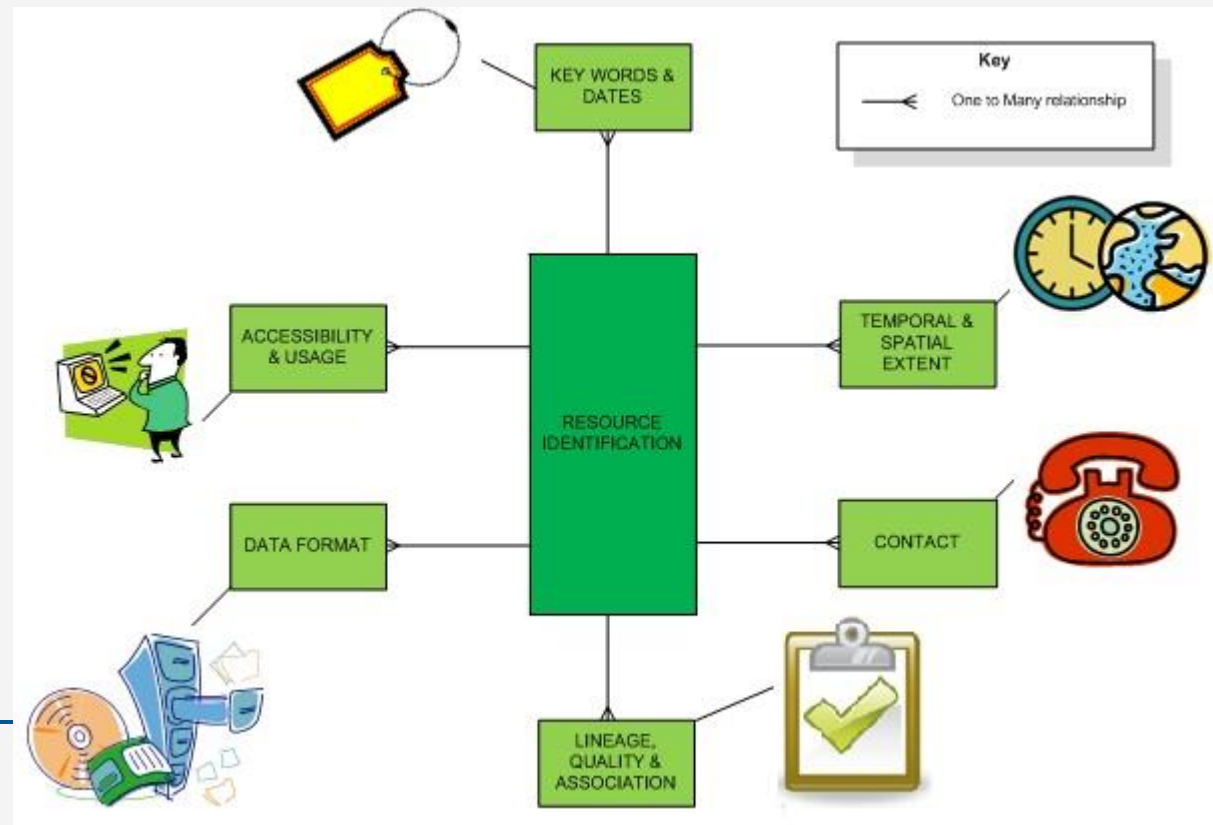
---

\* OGR is not fully compliant with the OpenGIS Simple Feature specification and is not approved as a reference implementation so the name was changed to OGR Simple Features Library

# Why use gdal and ogr?

For conversion between different geodata file formats

For quick info on metadata without a GIS



source:  
[http://www.earthdatamodels.org/designs/metadata\\_BGS.html](http://www.earthdatamodels.org/designs/metadata_BGS.html)

# Drivers

You need to have an appropriate driver that tells OGR how to read your data

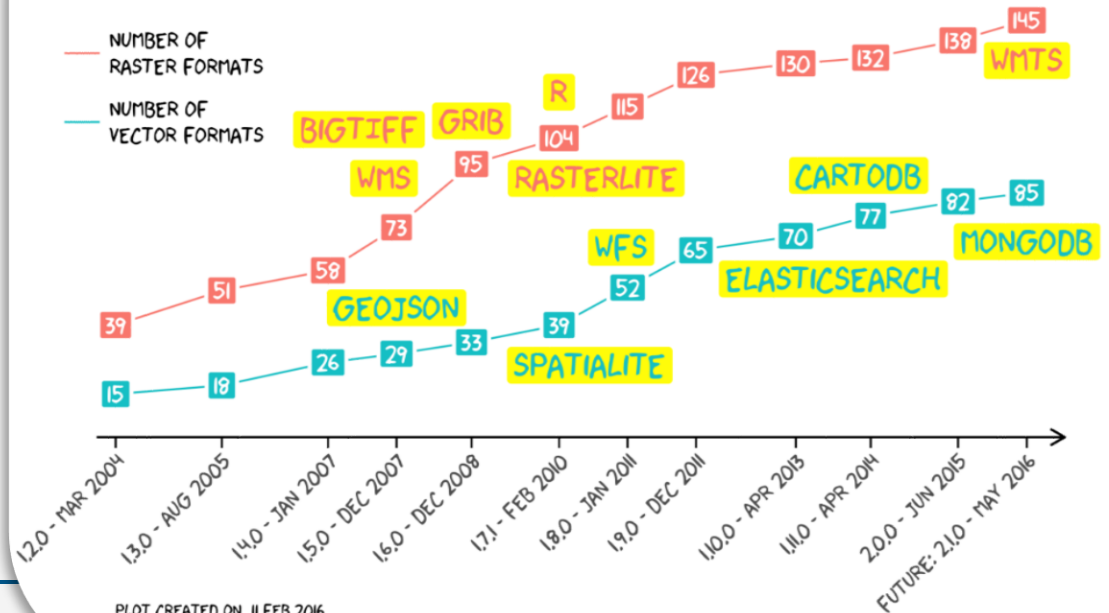
If no driver for a format is available, then OGR can't work with it

> 85 vector drivers available

> 145 raster drivers available

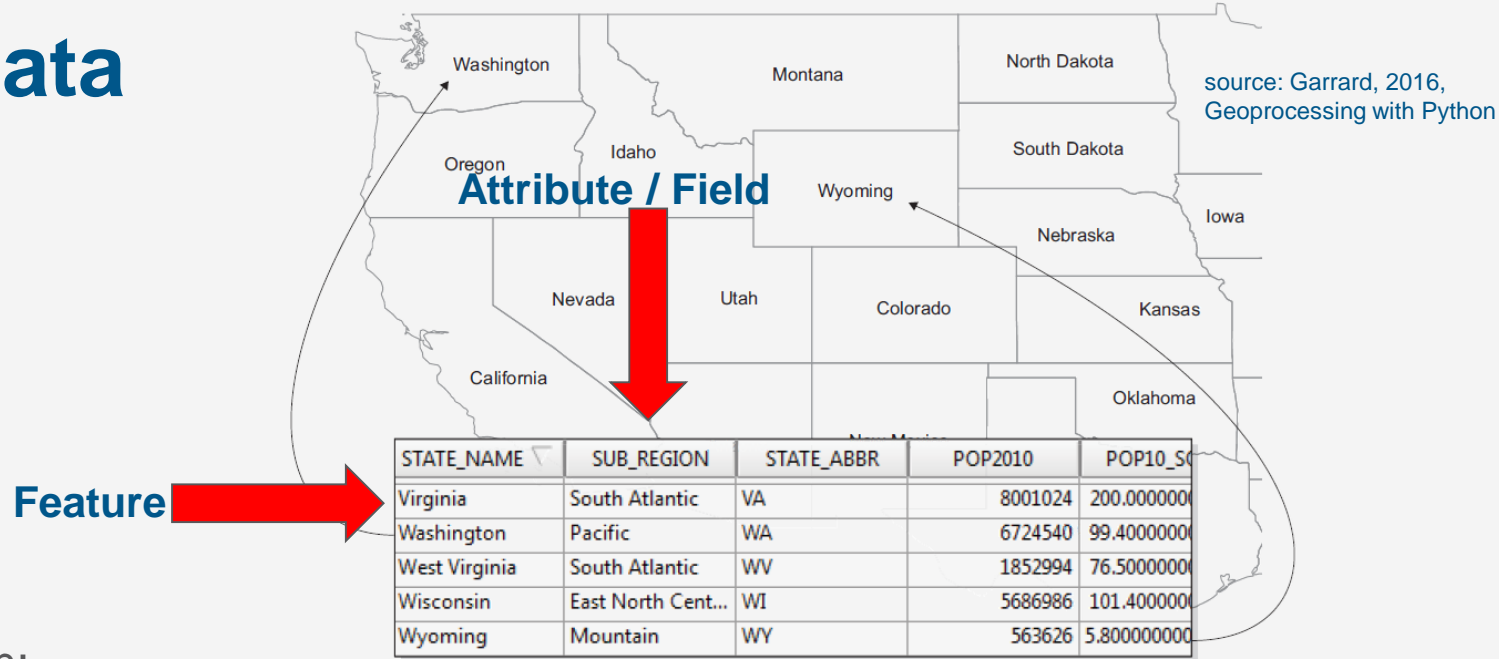
source: Teske, 2016,  
<https://github.com/dnltsk/gdal-ogr-format-driver-growth>

## GDAL-OGR: CONTINUES GROWTH OF FORMAT DRIVERS





# Vector data



Short recollection:

In *vector data* geographic features are represented as discrete geometries, specifically, points, lines, and polygons.

Geographic features with distinct boundaries, such as cities, work well as vector data, but continuous data, such as elevation, don't.

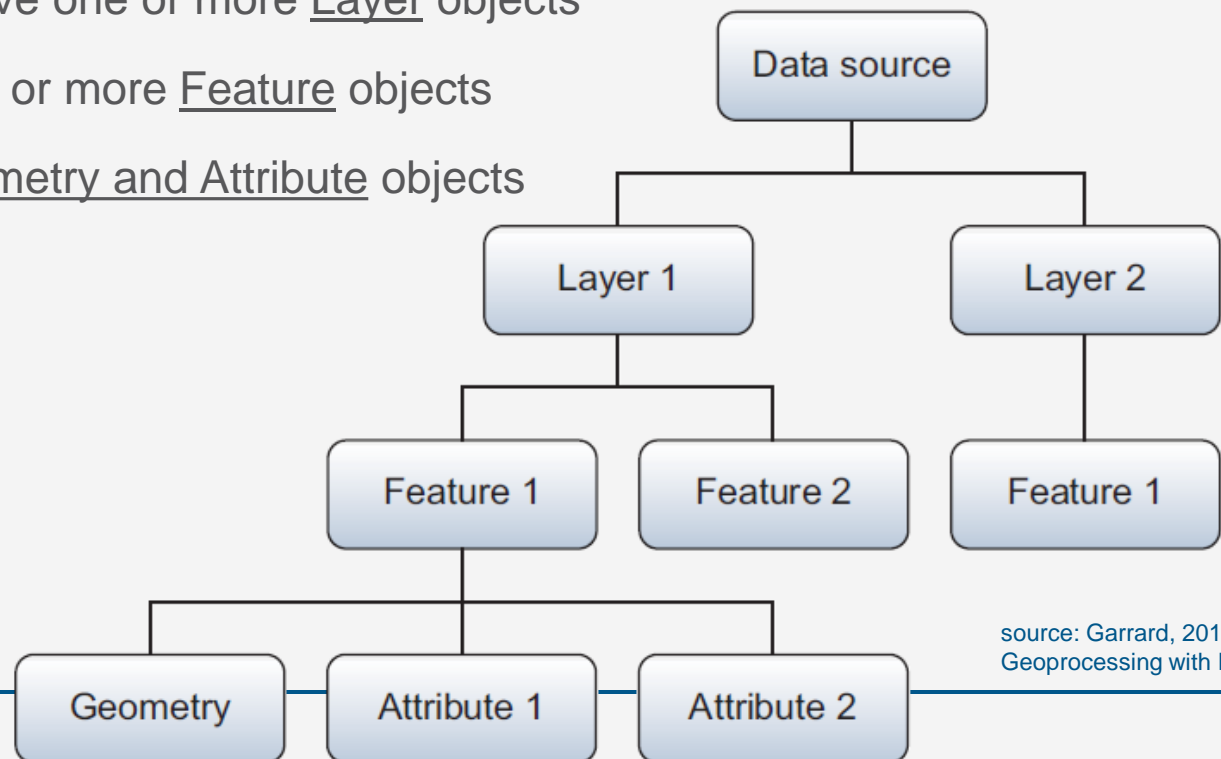
Each feature in vector data has associated attributes.



# IMPORTANT: OGR class structure

Understand how various objects in OGR are related to each other:

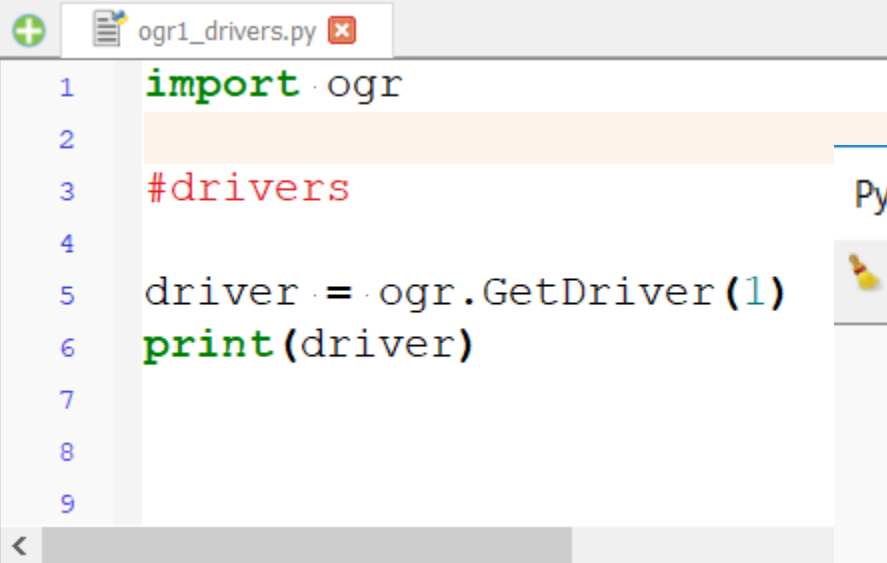
- When you open a file (e.g. shapefile), you have a DataSource object
- Data source can have one or more Layer objects
- Layer can have one or more Feature objects
- Features have Geometry and Attribute objects



source: Garrard, 2016,  
Geoprocessing with Python

# Checking available drivers in Python (1)

Can access drivers by number, but is not really useful ...



```
1 import ogr
2
3 #drivers
4
5 driver = ogr.GetDriver(1)
6 print(driver)
7
8
9
```

Unless you want to see them all ...

## Python Console



```
1 Python Console
2 Use iface to access QGIS API interface or Type help(iface) for more info
3 >>> exec(open('C:/Users/verstege/Documents/education/python_in_GIS/2017_2018/scripts/4_OGR1/ogr1_drivers.py').encode('utf-8')).read()
4 <osgeo.ogr.Driver; proxy of <Swig Object of type 'OGRDriverShadow *' at 0x000001705DF69B70> >
```

```
>>>
```

# Checking available drivers in Python (2)

```
ogr1_drivers.py
1  import ogr
2
3  #drivers
4  driver = ogr.GetDriver(1)
5  print(driver)
6
7  cnt = ogr.GetDriverCount()
8  formats_list = [] # Empty List
9
10 - for i in range(cnt):
11     driver = ogr.GetDriver(i)
12     driver_name = driver.GetName()
13 -     if not driver_name in formats_list:
14         formats_list.append(driver_name)
15
16 formats_list.sort() # Sorting the list of drivers
17
18 - for i in formats_list:
19     print(i)
```

## Python Console



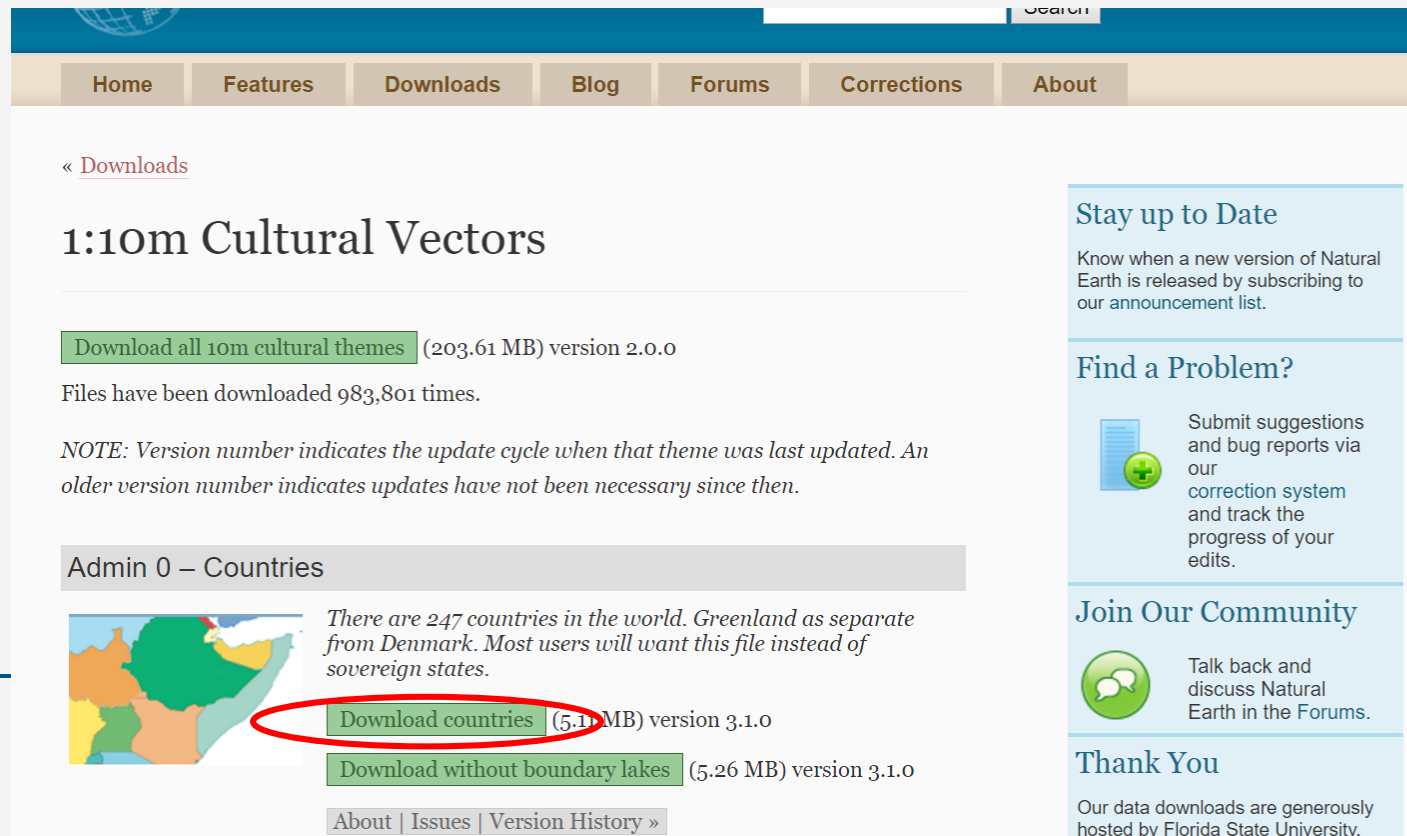
```
66 REC
67 S57
68 SEGUKOOA
69 SEGY
70 SOSI
71 SQLite
72 SUA
73 SVG
74 SXF
75 Selafin
76 TIGER
77 UK .NTF
78 VDV
79 VFK
80 WAsP
81 WFS
82 Walk
83 XLS
84 XLSX
85 XPlane
86 netCDF
87
```

90 drivers!

# Read a shapefile and access metadata (1)

For the next examples, I use the shapefiles of Admin 0 countries and populated places at: <http://www.naturalearthdata.com/downloads/10m-cultural-vectors/>

Or Google “populated places shapefile” to get to the website



« [Downloads](#)


## 1:10m Cultural Vectors

[Download all 10m cultural themes](#) (203.61 MB) version 2.0.0

Files have been downloaded 983,801 times.

*NOTE: Version number indicates the update cycle when that theme was last updated. An older version number indicates updates have not been necessary since then.*

### Admin 0 – Countries



There are 247 countries in the world. Greenland as separate from Denmark. Most users will want this file instead of sovereign states.

[Download countries](#) (5.1 MB) version 3.1.0

[Download without boundary lakes](#) (5.26 MB) version 3.1.0

[About](#) | [Issues](#) | [Version History](#) »

#### Stay up to Date

Know when a new version of Natural Earth is released by subscribing to our [announcement list](#).

#### Find a Problem?

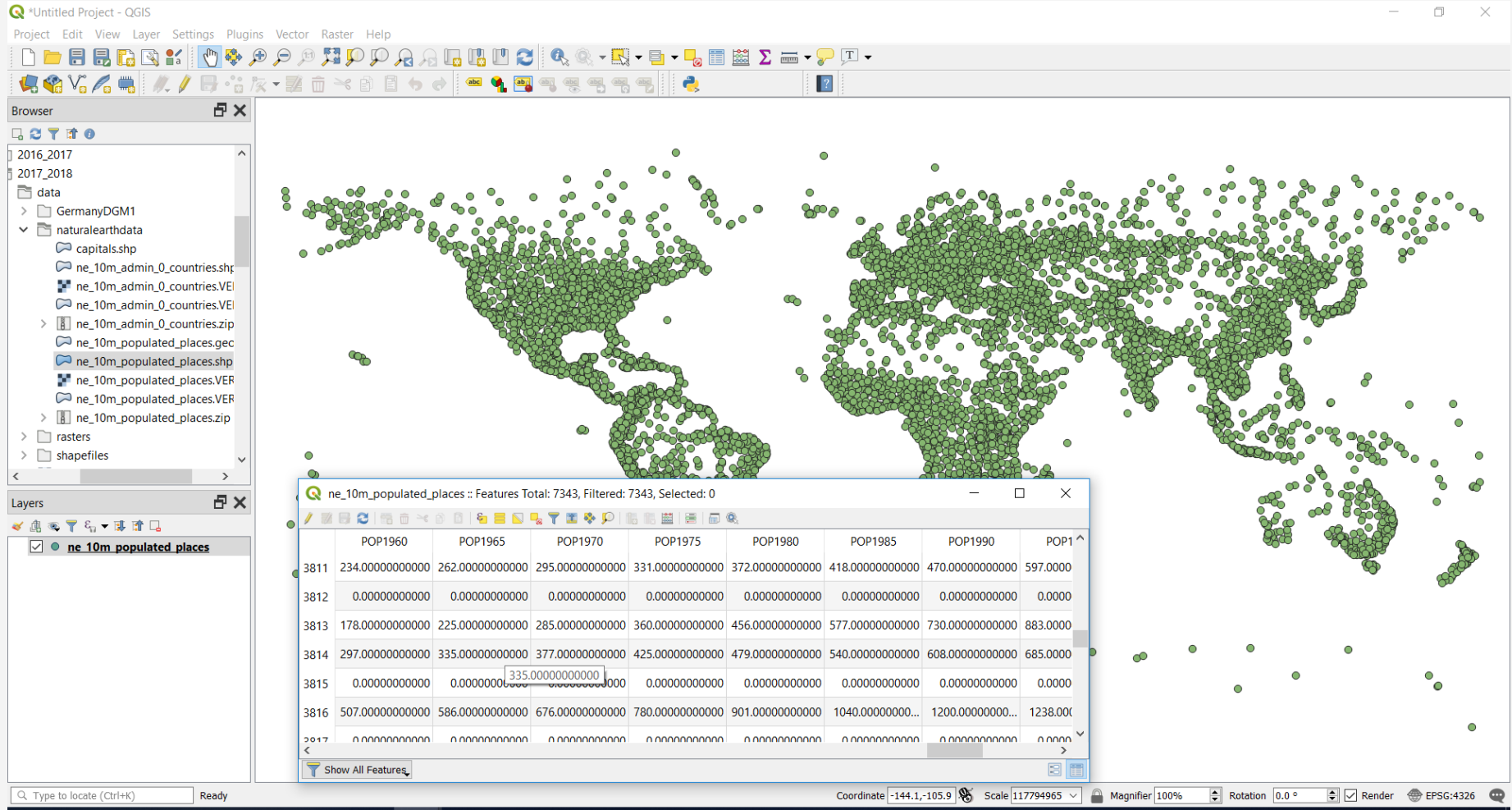
Submit suggestions and bug reports via our [correction system](#) and track the progress of your edits.

#### Join Our Community

Talk back and discuss Natural Earth in the [Forums](#).

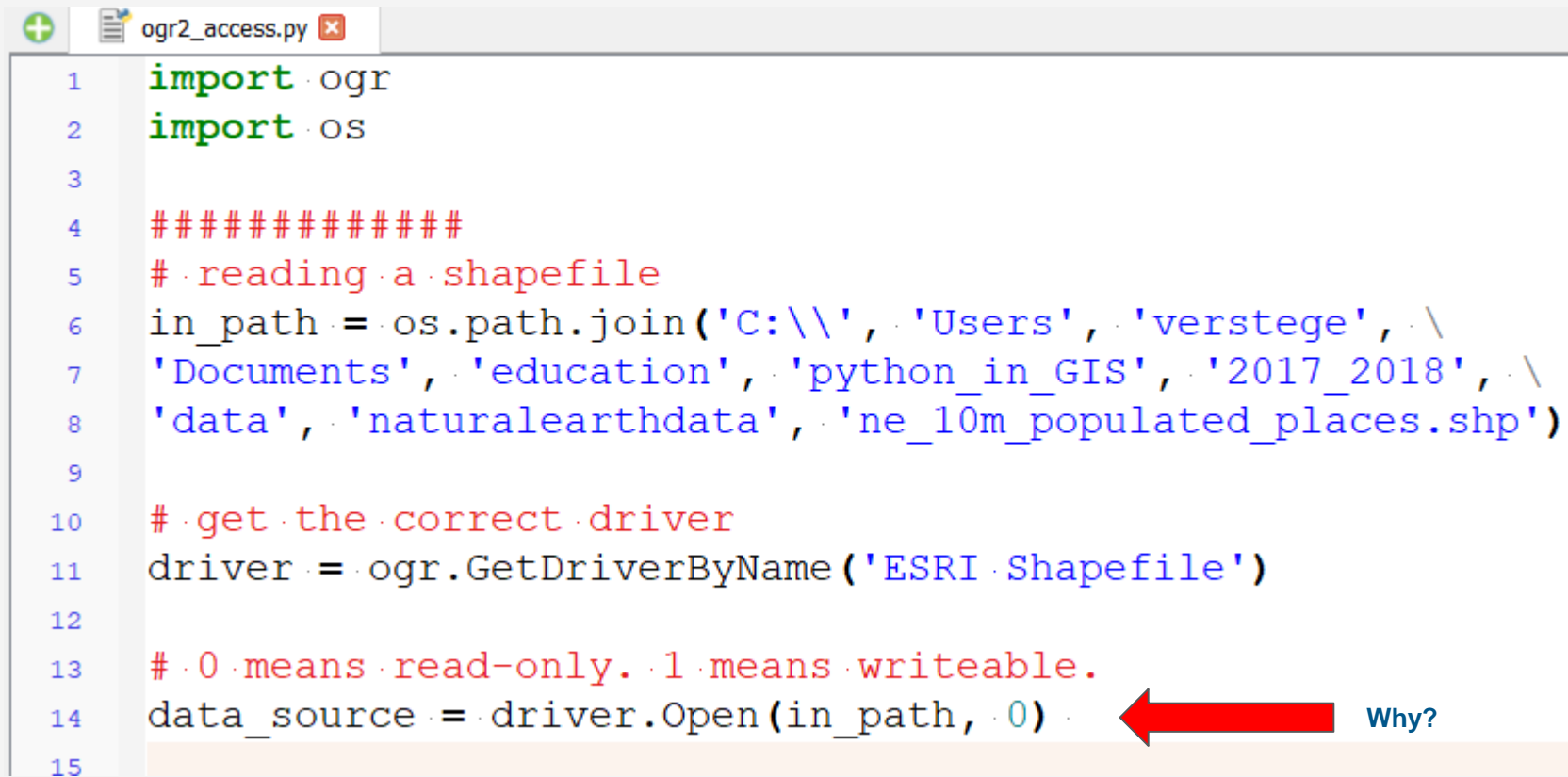
#### Thank You

Our data downloads are generously hosted by Florida State University.



# Read a shapefile and access metadata (2)

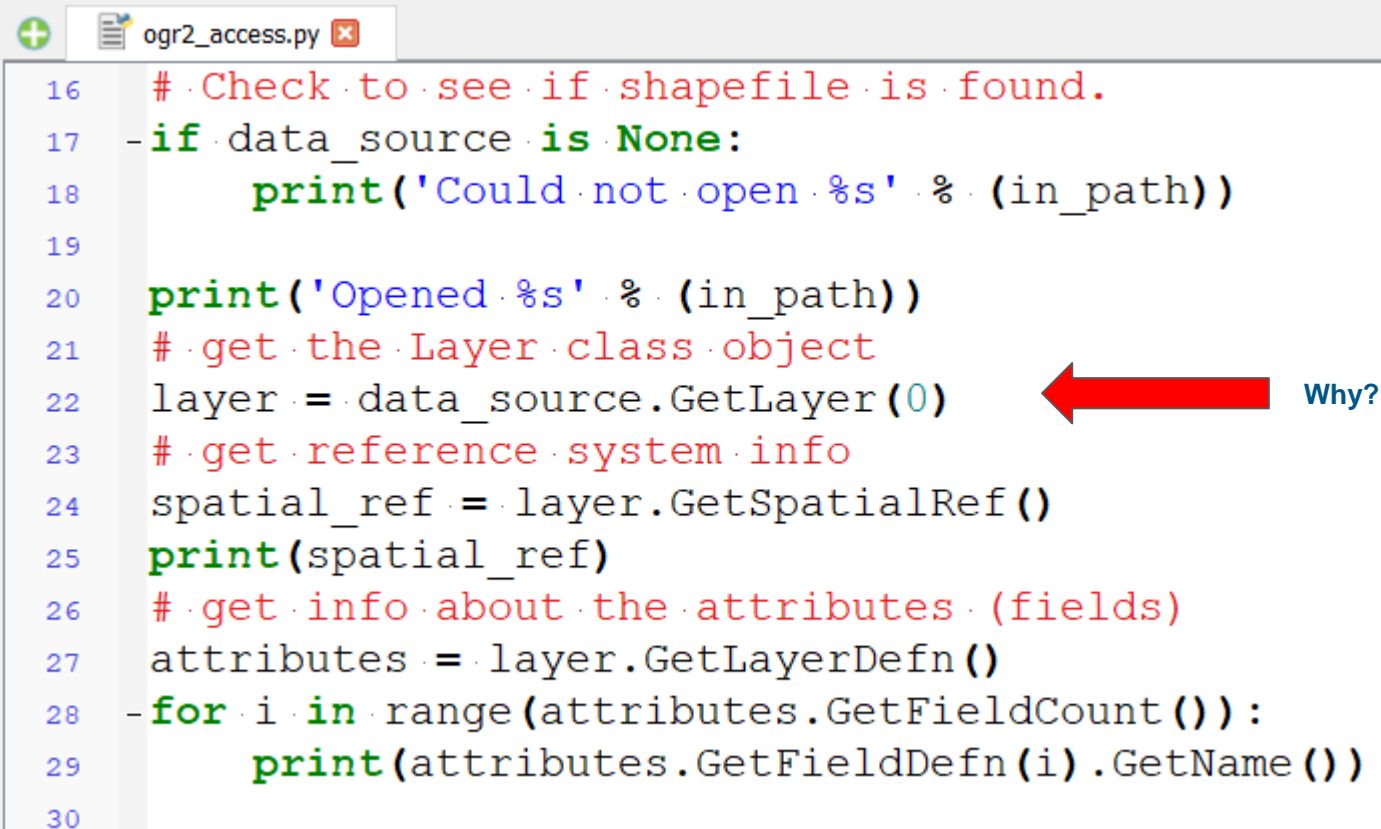
Import libraries, construct path to shapefile with os, and get the driver



```
1 import ogr
2 import os
3
4 #####
5 # reading a shapefile
6 in_path = os.path.join('C:\\', 'Users', 'verstege', '\\
7 'Documents', 'education', 'python_in_GIS', '2017_2018', '\\
8 'data', 'naturalearthdata', 'ne_10m_populated_places.shp')
9
10 # get the correct driver
11 driver = ogr.GetDriverByName('ESRI Shapefile')
12
13 # 0 means read-only. 1 means writeable.
14 data_source = driver.Open(in_path, 0)
15
```

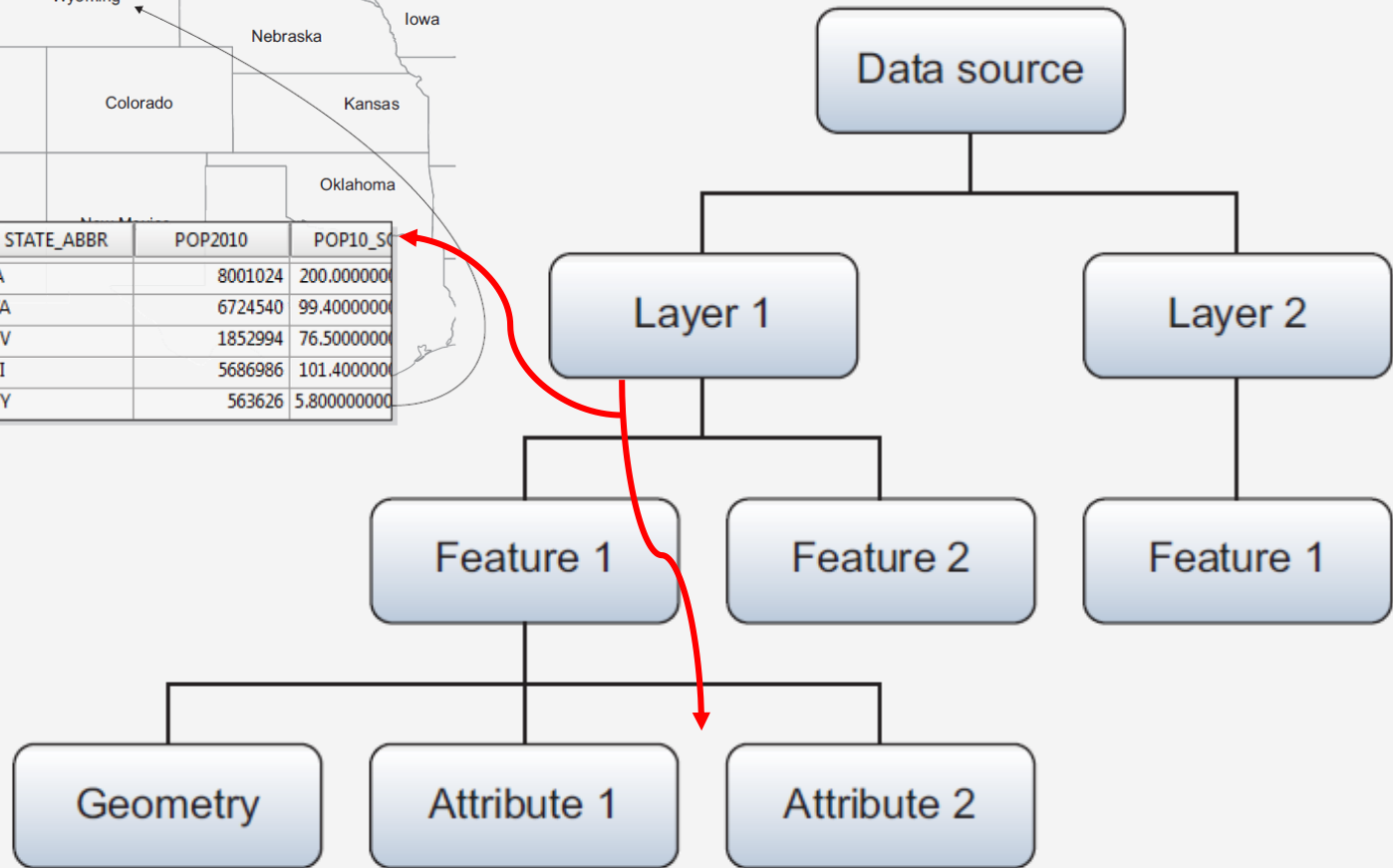
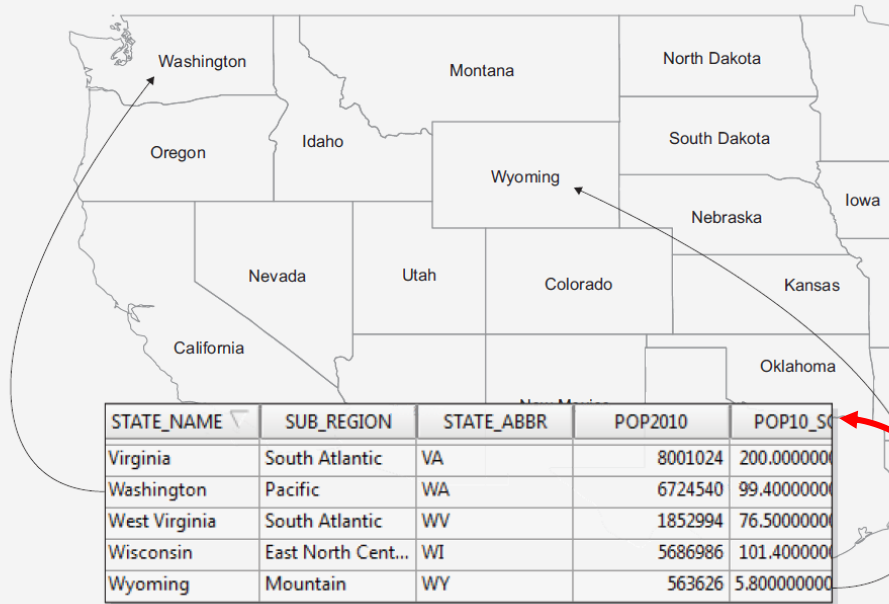
# Read a shapefile and access metadata (3)

Open the file to get the object of the DataSource class and check: **spatial reference info and attributes (field names)**



```
16 # Check to see if shapefile is found.
17 -if data_source is None:
18     print('Could not open %s' % (in_path))
19
20 print('Opened %s' % (in_path))
21 # get the Layer class object
22 layer = data_source.GetLayer(0)
23 # get reference system info
24 spatial_ref = layer.GetSpatialRef()
25 print(spatial_ref)
26 # get info about the attributes (fields)
27 attributes = layer.GetLayerDefn()
28 -for i in range(attributes.GetFieldCount()):
29     print(attributes.GetFieldDefn(i).GetName())
30
```





# Read a shapefile and access metadata (4)

What is the format of the reference system info?

WKT - Well-Known Text

```
1 Python Console
2 Use iface to access QGIS API i
  nterface or Type help(iface) f
  or more info
3 >>> exec(open('C:/Users/verste
  ge/Documents/education/python_
  in_GIS/2017_2018/scripts/4_OGR
  1/ogr2_access.py'.encode('utf-
  8')).read())
4 Opened C:\Users\verstege\Docum
  ents\education\python_in_GIS\2
  017_2018\data\naturalearthdata
  \ne_10m_populated_places.shp
5 GEOGCS["GCS_WGS_1984",
6     DATUM["WGS_1984",
7     SPHEROID["WGS_84", 6378
```

```
>>>
```

Python Console

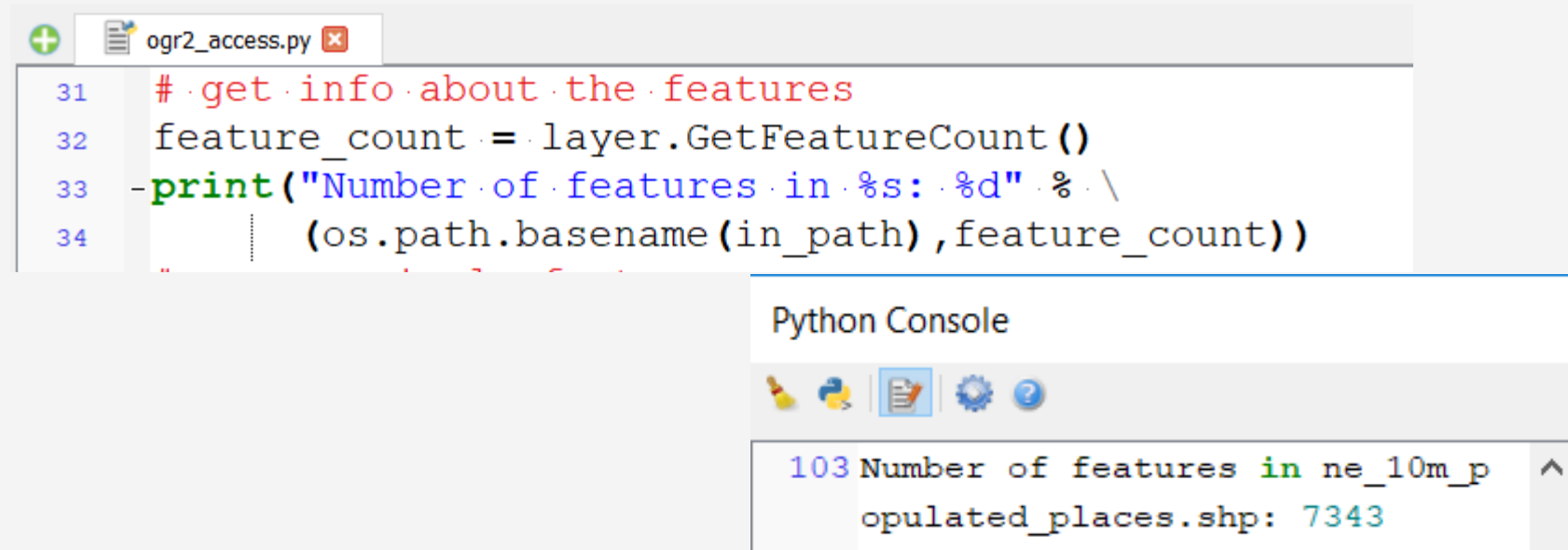


```
87 POP1960
88 POP1965
89 POP1970
90 POP1975
91 POP1980
92 POP1985
93 POP1990
94 POP1995
95 POP2000
96 POP2005
97 POP2010
98 POP2015
99 POP2020
100 POP2025
101 POP2050
102 CITYALT
```

```
>>>
```

# Read a shapefile and access metadata (5)

Open the file to get the object of the `DataSource` class and check: **number of features**



```
31 # get info about the features
32 feature_count = layer.GetFeatureCount()
33 - print("Number of features in %s: %d" % \
34       (os.path.basename(in_path), feature_count))
```


Python Console

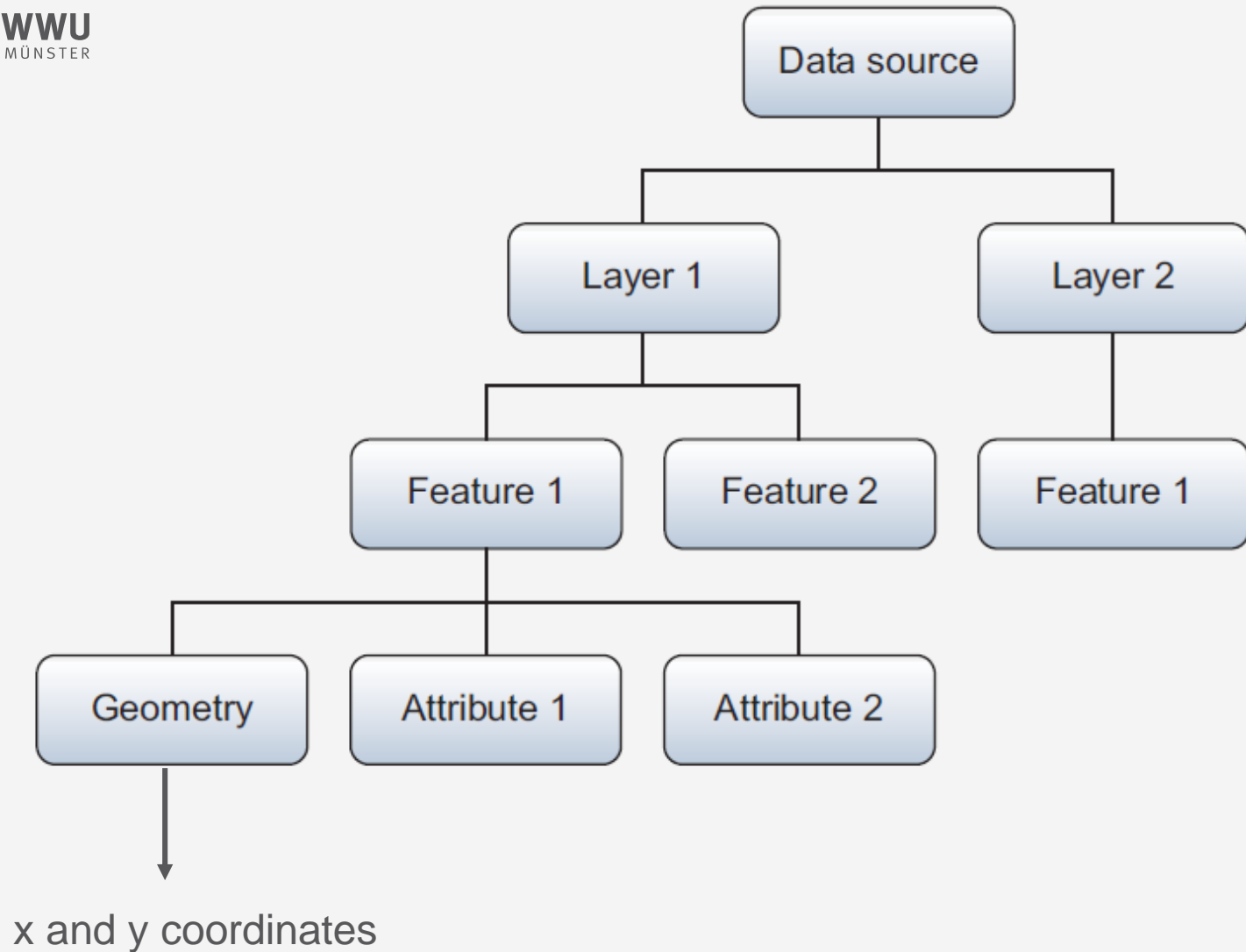
```
103 Number of features in ne_10m_p
    opulated_places.shp: 7343
```

# Accessing features and attributes (1)

Get information about **feature's geometry and attribute values**

```
ogr2_access.py
31 # get info about the features
32 feature_count = layer.GetFeatureCount()
33 - print("Number of features in %s: %d" % \
34       (os.path.basename(in_path), feature_count))
35 # access single features
36 - for feat in layer:
37     pt = feat.geometry()
38     x = pt.GetX()
39     y = pt.GetY()
40     name = feat.GetField('NAME')
41     pop = feat.GetField('POP_MAX')
42     country = feat.GetField('ADM0NAME')
43     if country == 'Germany': print(name, pop, x, y)
44
```





# Accessing features and attributes (2)

Python Console



```
103 Number of features in ne_10m_p
    opulated_places.shp: 7343
104 Mainz 184997 8.273219155500556
    49.98247245501278
105 Schwerin 96641 11.416698610531
    51.2 53.633304077430296
106 Bielefeld 331906 8.53001135115
    8521 52.02998821930004
107 Dortmund 588462 7.450025592690
    793 51.52996706044394
108 Duisburg 1276757 6.75001664086
    5056 51.429973163959176
109 Wuppertal 776525 7.16999100610
    0929 51.250009988502654
110 Essen 1742135 7.01661535505888
    9 51.44999778147235
```

# Accessing features and attributes (3)

The `GetField()` function returns data in same data type as in underlying dataset

If you want the data in another format, use format specific functions, such as `GetFieldAsString`, e.g.:

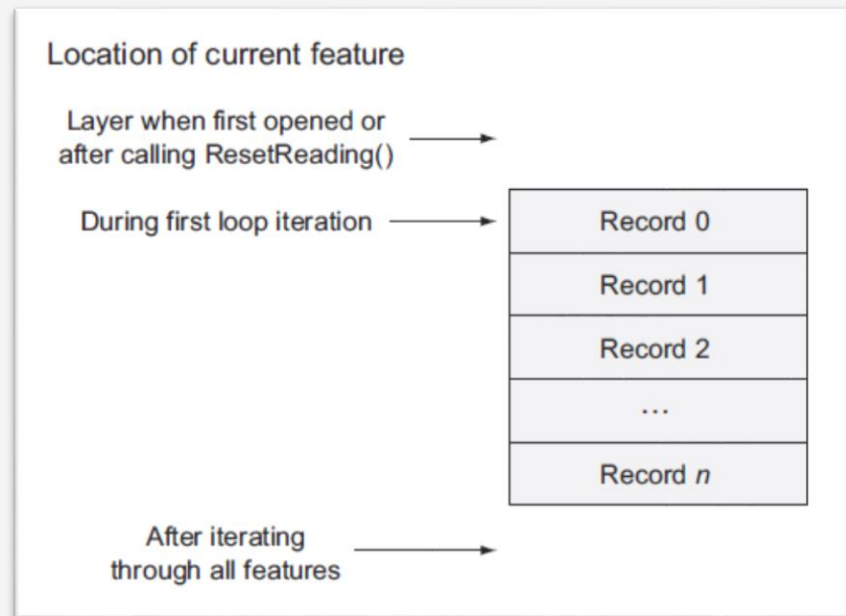
```
pop = feat.GetFieldAsString('POP_MAX')
```

```
35 # access single features
36 for feat in layer:
37     pt = feat.geometry()
38     x = pt.GetX()
39     y = pt.GetY()
40     name = feat.GetField('NAME')
41     pop = feat.GetField('POP_MAX')
42     country = feat.GetField('ADM0NAME')
43     if country == 'Germany': print(name, pop, x, y)
44
```

# Accessing features and attributes (4)

Note that ogr keeps track of which feature was last accessed

If you have accessed all features and want to access them again, use the `layer.ResetReading()` function





# Exercise #1

In the data folder there is a shapefile `gps_track_projected.shp`

Write a script, using `ogr`, to:

- open this file
- print the number of features
- loop over all features
- print the elevation (attribute 'ele') of each feature

# Attribute filters (1)

You now know how to iterate through features with ogr and select features with a particular attribute value. But there is an easier way: attribute filters

To set an attribute filter, you need a conditional statement much like the WHERE clause in an SQL statement.

Attribute filter applies to layer: `lyr.SetAttributeFilter('continent="Asia"')`

Use the standard logical operators, such as `=`, `!=`, `<>`, `>`, `<`, `>=`, and `<=`

```
'Population < 50000'  
'Population >= 25000'  
'Type_code != 7'  
'Name = "Cairo"  
"Name = 'Moscow'"  
'Name != "Tokyo"'
```

 **Safer!!**

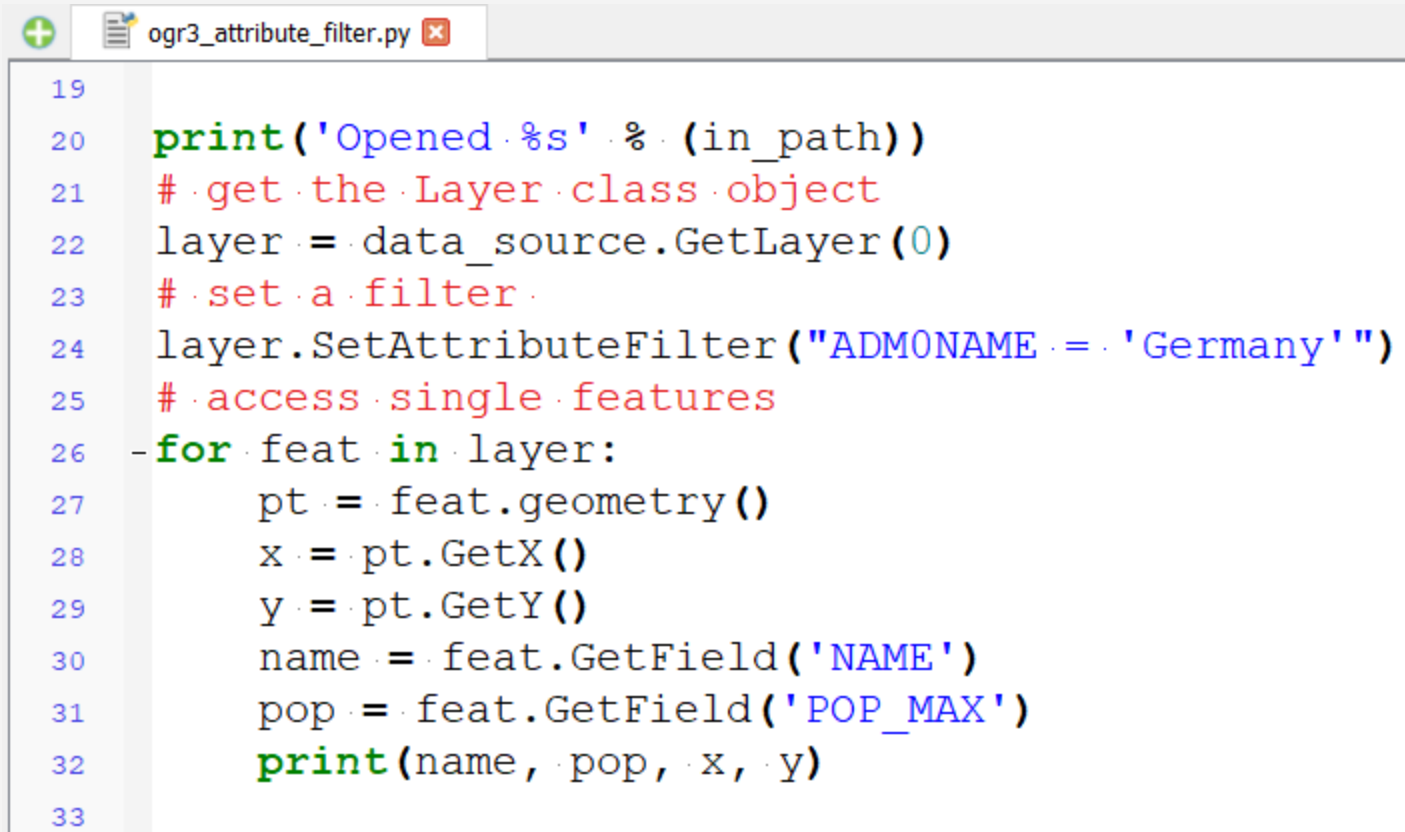
**Numeric comparisons don't  
require quotes around the number**

**String comparisons require  
either single or double quotes**

source: Garrard, 2016,  
Geoprocessing with Python

# Attribute filters (2)

So, we can do the same as before with a filter:



```
19
20 print('Opened %s' % (in_path))
21 # get the Layer class object
22 layer = data_source.GetLayer(0)
23 # set a filter
24 layer.SetAttributeFilter("ADM0NAME = 'Germany'")
25 # access single features
26 - for feat in layer:
27     pt = feat.geometry()
28     x = pt.GetX()
29     y = pt.GetY()
30     name = feat.GetField('NAME')
31     pop = feat.GetField('POP_MAX')
32     print(name, pop, x, y)
33
```

# Attribute filters (4)

Some more conditions you can use:

- Conditions can be negated using NOT, and NULL is used to indicate a null or no data value in the attribute table:
  - **'(Population < 50000) OR NOT (Place\_type = "County Seat")'**
  - **'County NOT NULL'**
- Ranges can be selected with BETWEEN or IN:
  - **'Population BETWEEN 25000 AND 50000'**
  - *This is the same as:* **'(Population > 25000) AND (Population < 50000)'**
  - **'Type\_code IN (4, 3, 7)'**
  - *This is the same as:* **'(Type\_code = 4) OR (Type\_code = 3) OR (Type\_code = 7)'**

# Spatial filters (1)

Spatial filters let you limit the features by spatial extent rather than by attribute value

Again, assume that you want to look at German cities only, but that there is no attribute (field) indicating the country name

For that, we'll use the other shapefile we've downloaded, of country polygons

QGIS: \*Untitled Project - QGIS

Project Edit View Layer Settings Plugins Vector Raster Help

**Browser**

- 2016\_2017
- 2017\_2018
- data
  - GermanyDGM1
  - naturalearthdata
    - capitals.shp
    - ne\_10m\_admin\_0\_countries
    - ne\_10m\_admin\_0\_countries
    - ne\_10m\_admin\_0\_countries
    - ne\_10m\_admin\_0\_countries
    - ne\_10m\_populated\_places
    - ne\_10m\_populated\_places
    - ne\_10m\_populated\_places
    - ne\_10m\_populated\_places
    - ne\_10m\_populated\_places
  - rasters
  - shapefiles

**Layers**

- ☒ ne\_10m\_admin\_0\_countries
- ☐ ne\_10m\_populated\_places

**ne\_10m\_admin\_0\_countries : Features Total: 255, Filtered: 255, Selected: 0**

	scalerank	featurecla	LABELRANK	SOVEREIGNT	SOV_A3	ADM0_DIF	LEVEL	TYPE
1	0	Admin-0 country	2.00	Argentina	ARG	0.00	2.00	Sovereign
2	0	Admin-0 country	6.00	Armenia	ARM	0.00	2.00	Sovereign
3	0	Admin-0 country	6.00	Andorra	AND	0.00	2.00	Sovereign
4	0	Admin-0 country	4.00	United Arab Em...	ARE	0.00	2.00	Sovereign
5	0	Admin-0 country	6.00	Albania	ALB	0.00	2.00	Sovereign
6	3	Admin-0 country	6.00	Finland	FIN	1.00	2.00	Country
7	0	Admin-0 country	3.00	Angola	AGO	0.00	2.00	Sovereign

Show All Features

Coordinate: -152.1,-14.8 Scale: 117794965 Magnifier: 100% Rotation: 0.0° Render EPSG:4326

## Spatial filters (2)

ogr4\_spatial\_filter.py

```
1  import ogr
2  import os
3
4  #####
5  # reading the places shapefile
6  in_path = os.path.join('C:\\', 'Users', 'verstege', '\\
7  'Documents', 'education', 'python_in_GIS', '2017_2018', '\\
8  'data', 'naturalearthdata', 'ne_10m_populated_places.shp')
9
10 # reading the countries shapefile
11 in_path2 = os.path.join('C:\\', 'Users', 'verstege', '\\
12 'Documents', 'education', 'python_in_GIS', '2017_2018', '\\
13 'data', 'naturalearthdata', 'ne_10m_admin_0_countries.shp')
14
15 # get the correct driver
16 driver = ogr.GetDriverByName('ESRI Shapefile')
17
18 # 0 means read-only. 1 means writeable.
19 places_source = driver.Open(in_path, 0)
20 countries_source = driver.Open(in_path2, 0)
```

Copy the previous script, and now load two shapefiles

# Spatial filters (3)

Select the polygon of Germany and save the geometry in a variable

```
ogr4_spatial_filter.py
25
26 print('Opened %s' % (in_path2))
27 # get the Layer class objects
28 places_layer = places_source.GetLayer(0)
29 countries_layer = countries_source.GetLayer(0)
30 # set a filter to the countries layer
31 countries_layer.SetAttributeFilter("NAME = 'Germany'")
32 # get the feature (should be only one!)
33 feature_count = countries_layer.GetFeatureCount()
34 - print("Number of features in %s: %d" % \
35         (os.path.basename(in_path2), feature_count))
36 feat = countries_layer.GetNextFeature()
37 germany = feat.geometry().Clone()
38 print(germany)
```

← Attribute filter

← Clone geometry



## Python Console



```
5 Number of features in ne_10m_admin_0_countries.shp: 1
6 MULTIPOLYGON (((6.742198113000
11 53.5783552100001,6.74952233
200011 53.572414455,6.75652103
00001 53.562892971,6.747569207
00008 53.5659854190001,6.73462
975400005 53.5751813820001,6.7
2608483200014 53.5771345070001
,6.71599368600008 53.576076565
0001,6.70118248800011 53.57143
78930001,6.69507897200012 53.5
702985700001,6.67709394600007
53.5756289730001,6.66334069100
014 53.5873477230001,6.6595158
2100013 53.5991071640001,6.671
```

```
>>>
```

# Spatial filters (4)

Apply the spatial filter with `places_layer.SetSpatialFilter(germany)` and loop through the features of the places layer to check your selection


```
ogr4_spatial_filter.py
38 print(germany)
39
40 # now filter the features in the places layer
41 places_layer.SetSpatialFilter(germany)
42 # access single features in the places layer
43 for feat in places_layer:
44     pt = feat.geometry()
45     x = pt.GetX()
46     y = pt.GetY()
47     name = feat.GetField('NAME')
48     print(name, x, y)
49
```

 Spatial filter

# Spatial filters (5)

Result

Python Console



```
58 Cologne 6.948058575221637 50.9
    3194954014825
59 Dresden 13.750002806257669 51.
    04997051910084
60 Frankfurt 8.675015420169643 50
    .09997682606331
61 Hamburg 9.998053285520314 53.5
    5197049556244
62 Munich 11.573047588912061 48.1
    31887894629244
63 Berlin 13.399602764700546 52.5
    23764522251156
64
```

```
>>>
```

# Writing vector data to a different file format

ogr5\_write.py To save all data in file to a new format: CopyDataSource().

```
1 import ogr
2 import os
3
4 # reading a shapefile
5 in_path = os.path.join('C:\\', 'Users', 'verstege', '\\
6 'Documents', 'education', 'python_in_GIS', '2017_2018', '\\
7 'data', 'naturalearthdata', 'ne_10m_populated_places.shp')
8
9 # get the correct driver
10 in_driver = ogr.GetDriverByName('ESRI Shapefile')
11 places_source = in_driver.Open(in_path, 0)
12
13 # write the data to a geojson
14 out_file = os.path.join('C:\\', 'Users', 'verstege', '\\
15 'Documents', 'education', 'python_in_GIS', '2017_2018', '\\
16 'data', 'naturalearthdata', 'ne_10m_populated_places.geojson')
17 out_driver = ogr.GetDriverByName('GeoJSON')
18 out_ds = out_driver.CopyDataSource(places_source, out_file)
19 del out_ds
```

Choose output type

To release the file

ne\_10m\_populated\_places.geojson - WordPad

File Home View

Paste Cut Copy

Courier New 11

**B** *I* U abc  $\times_2$   $\times^2$  A

Paragraph

Picture Paint drawing Date and time Insert object

Find Replace Select all

Editing

Clipboard Font Paragraph Insert Editing

```
{
  "type": "FeatureCollection",
  "name": "ne_10m_populated_places",
  "crs": { "type": "name", "properties": { "name":
    "urn:ogc:def:crs:OGC:1.3:CRS84" } },
  "features": [
    { "type": "Feature", "id": 0, "properties": { "SCALERANK": 10,
      "NATSCALE": 1, "LABELRANK": 8, "FEATURECLA": "Admin-1 capital",
      "NAME": "Colonia del Sacramento", "NAMEPAR": null, "NAMEALT":
      null, "DIFFASCII": 0, "NAMEASCII": "Colonia del Sacramento",
      "ADM0CAP": 0.0, "CAPALT": 0.0, "CAPIN": null, "WORLD_CITY": 0.0,
      "MEGACITY": 0, "SOV0NAME": "Uruguay", "SOV_A3": "URY",
      "ADM0NAME": "Uruguay", "ADM0_A3": "URY", "ADM1NAME": "Colonia",
      "ISO_A2": "UY", "NOTE": null, "LATITUDE": -34.479999005400003,
      "LONGITUDE": -57.840002473399998, "CHANGED": 4.0, "NAMEDIFF": 1,
      "DIFFNOTE": "Added missing admin-1 capital. Population from
      GeoNames.", "POP_MAX": 21714, "POP_MIN": 21714, "POP_OTHER": 0,
      "RANK_MAX": 7, "RANK_MIN": 7, "GEONAMEID": 3443013.0, "MEGANAME":
      null, "LS_NAME": null, "LS_MATCH": 0, "CHECKME": 0, "MAX_POP10":
      0.0, "MAX_POP20": 0.0, "MAX_POP50": 0.0, "MAX_POP300": 0.0,
      "MAX_POP310": 0.0, "MAX_NATSCA": 0.0, "MIN_AREAKM": 0.0,
```

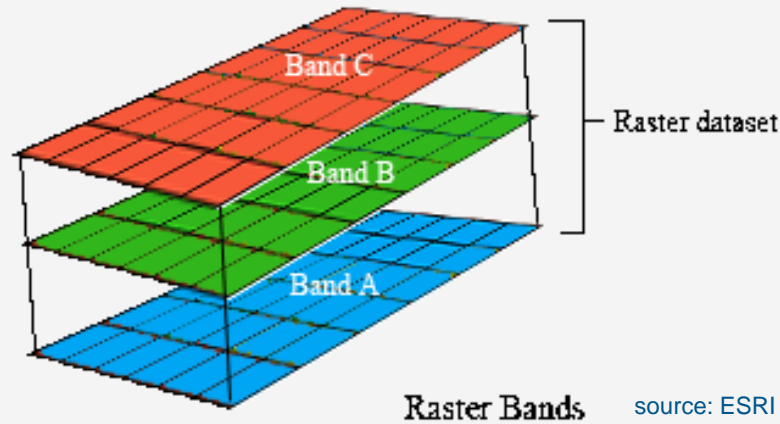
100%

## Exercise #2

Convert the shapefile you have to geojson using OGR in Python (in QGIS).



# Raster data



Short recollection:

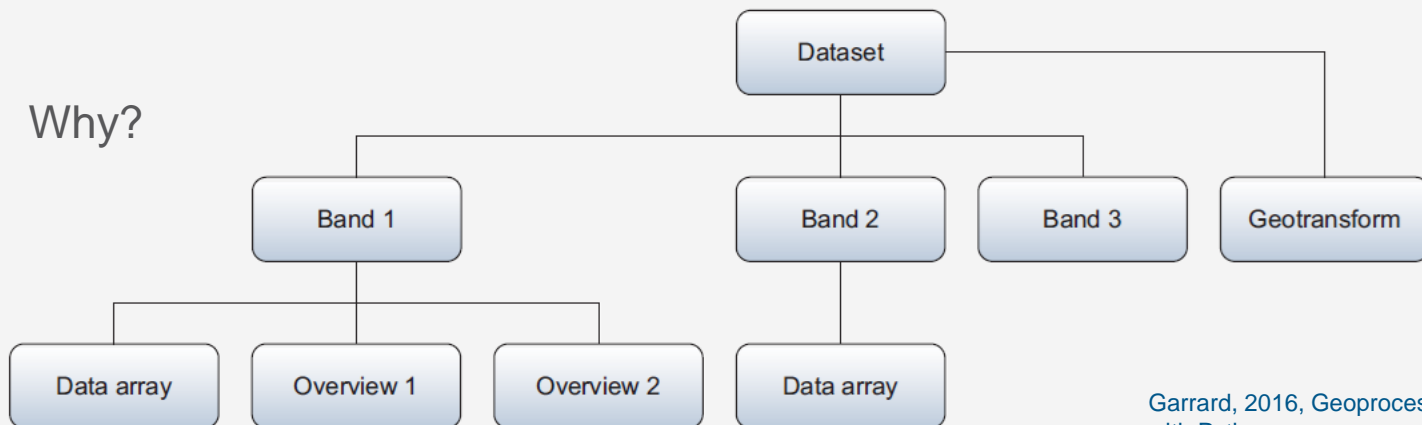
- In *raster data* geographic features are represented as surfaces, divided into a grid of equally sized cells.
- Geographic features that are continuous, such as elevation, work well as raster data, but discrete objects, such as houses, do not.
- Each band in raster data represents a single attribute



# GDAL class structure

Understand how various objects in GDAL are related to each other:

- When you open a file (e.g. geotiff), you have a DataSource object
- Data source can have one or more Band objects (instead of layers in vector data)
- Band contains the pixel data as Data array objects and possibly Overviews
- Geotransform is comparable to geometry in a vector, but at the Dataset level



Garrard, 2016, Geoprocessing  
with Python

# Reading a raster (1)

```
gdal0_read_raster.py
1  import gdal
2  import numpy as np
3  import os
4
5  #####
6  data_dir = os.path.join('C:\\', 'Users', 'verstege', '\\
7  'Documents', 'education', 'python_in_GIS', 'data')
8
9  # Path to the raster
10 in_path = os.path.join(data_dir, 'rasters', '\\
11     'clipped_dem.tif')
12
13 # Open the raster
14 rast_data_source = gdal.Open(in_path)
15
```

Import the gdal library

Create path to a raster

Open without specifying the driver

# Reading a raster (2)

Notice no parentheses, because they're class variables, not methods / functions

gdal0\_read\_raster.py

```
15
16 # Get metadata at data_source level
17 print('Nr of bands:', rast_data_source.RasterCount)
18 cols = rast_data_source.RasterXSize
19 rows = rast_data_source.RasterYSize
20 print('Size:', cols, rows)
21
22 # Select (the only) band and get meta data at band level
23 srcband = rast_data_source.GetRasterBand(1)
24 srcband.ComputeStatistics(0)
25 print('Minimum is:', srcband.GetMinimum())
26 print('Maximum is:', srcband.GetMaximum())
27
28 # Create empty array and catch the data in it
29 data = np.empty((rows, cols))
30 srcband.ReadAsArray(buf_obj=data)
31 print(data)
```

← Get meta data

← Band indices start at 1!!

← Reading data with buffer

# Reading a raster (3)

Result

Python Console



```
2 Use iface to access QGIS API interface or Type help(if
  ace) for more info
3 Security warning: typing commands from an untrusted so
  urce can lead to data loss and/or leak
4 >>> exec(open('C:/Users/verstege/Documents/workshops_c
  onferences/2019_2020/ILS_Python/materials/3_OGR_GDAL/g
  dal0_read_raster.py'.encode('utf-8')).read())
5 Nr of bands: 1
6 Size: 2463 1809
7 Minimum is: 32.0
8 Maximum is: 139.0
9 [[65. 64. 65. ... 64. 64. 64.]
10  [64. 64. 65. ... 64. 64. 64.]
11  [64. 65. 65. ... 64. 64. 64.]
12  ...
13  [73. 73. 73. ... 47. 47. 47.]
14  [73. 73. 73. ... 47. 47. 47.]
15  [73. 73. 73. ... 47. 47. 47.]]
16
```

# Creating a new raster and adding data (1)

```
gdal1_new_raster.py
1 import gdal
2 import numpy as np
3 import os
4 import osr
5
6 #####
7 data_dir = os.path.join('C:\\', 'Users', 'verstege', '\\
8 'Documents', 'education', 'python_in_GIS', '2017_2018', '\\
9 'data')
10 # Path to the new raster file
11 out_fn = os.path.join(data_dir, 'rasters', 'test.tif')
12 # Spatial reference system
13 srs = osr.SpatialReference()
14 srs.ImportFromEPSG(4326) # WGS84
15 # Create the data
16 data = np.arange(0, 100).reshape(10, 10)
```

← We need gdal, numpy, os and osr

← srs library for spatial reference info!

← Integers from 0 to 99  
In a 10x10 matrix

# Creating a new raster and adding data (2)

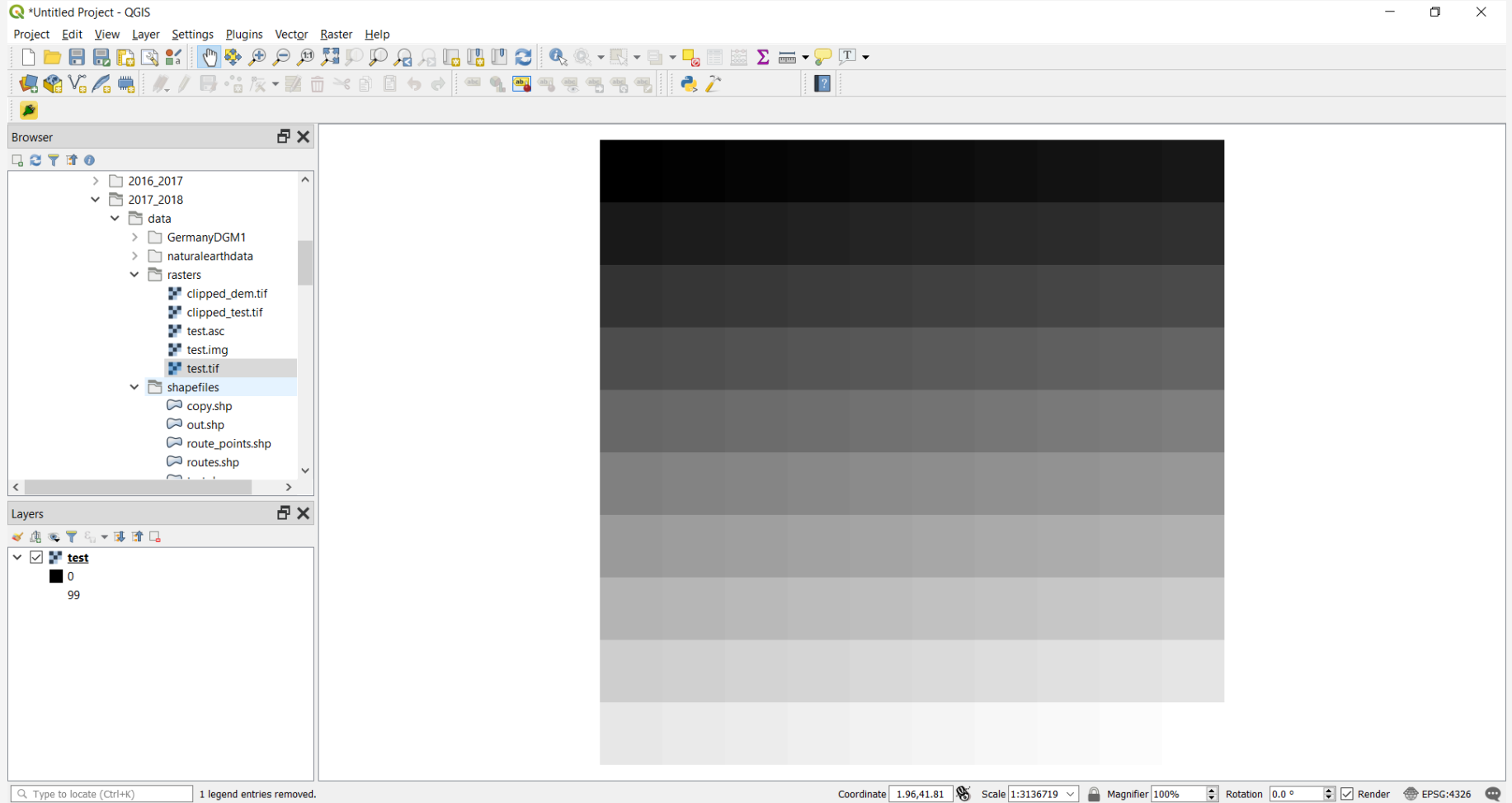
```
gdal1_new_raster.py
18 # Define properties of the new raster
19 originX = 4
20 originY = 52
21 pixelWidth = 1
22 pixelHeight = -1
23
24 # Create the output file and add the properties
25 driver = gdal.GetDriverByName('GTiff')
26 out_ds = driver.Create(out_fn, data.shape[1],
27                        data.shape[0], 1, gdal.GDT_UInt32)
28 out_ds.SetGeoTransform((originX, pixelWidth, 0, \
29                        originY, 0, pixelHeight))
30 out_ds.SetProjection(srs.ExportToWkt())
31
32 # Add the data
33 outband = out_ds.GetRasterBand(1)
34 outband.WriteArray(data)
35 out_ds.FlushCache()
36
```

columns, rows, bands, data type

spatial reference info as a tuple

add the NumPy array to the band

# Result



# Data type options

**Table 9.1 GDAL data type constants**

Constant	Data type
GDT_Unknown	Unknown
GDT_Byte	Unsigned 8-bit integer (byte)
GDT_UInt16	Unsigned 16-bit integer
GDT_Int16	Signed 16-bit integer
GDT_UInt32	Unsigned 32-bit integer
GDT_Int32	Signed 32-bit integer
GDT_Float32	32-bit floating point
GDT_Float64	64-bit floating point
GDT_CInt16	16-bit complex integer
GDT_CInt32	32-bit complex integer
GDT_CFloat32	32-bit complex floating point
GDT_CFloat64	64-bit complex floating point
GDT_TypeCount	Number of available data types

source: Garrard, 2016,  
Geoprocessing with Python

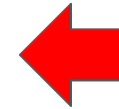


# Other data type, example (1)



gdal2\_edit\_data.py

```
13 # .Srs
14 srs = osr.SpatialReference()
15 srs.ImportFromEPSG(4326) # .WGS84
16 # .Create the data
17 -data = np.array([[1,1,1,1,1,1,1,1,1,1,1,1,1],
18                  [1,1,1,1,1,1,1,1,1,1,1,1,1],
19                  [1,0,1,0,0,0,1,0,0,0,1,0,1],
20                  [1,0,1,0,1,1,1,0,1,1,1,0,1],
21                  [1,0,1,0,0,0,1,0,1,0,1,0,1],
22                  [1,0,1,0,1,1,1,0,1,0,1,0,1],
23                  [1,0,1,0,1,1,1,0,0,0,1,0,1],
24                  [1,1,1,1,1,1,1,1,1,1,1,1,1],
25                  [1,1,1,1,1,1,1,1,1,1,1,1,1],
26                  [1,1,1,1,1,1,1,1,1,1,1,1,1]])
27 originX = 4
28 originY = 52
29 pixelWidth = 1
30 pixelHeight = -1
```



a binary array



Again, cell size in y-direction always negative

## Other data type, example (2)


```
gdal2_edit_data.py x
31
32 # Create the output file and set parameters
33 driver = gdal.GetDriverByName('GTiff')
34 out_ds = driver.Create(out_fn, data.shape[1],
35                        data.shape[0], 1, gdal.GDT_Byte)
36 out_ds.SetGeoTransform((originX, pixelWidth, 0, \
37                        originY, 0, pixelHeight))
38 out_ds.SetProjection(srs.ExportToWkt())
39
40 outband = out_ds.GetRasterBand(1)
41 outband.WriteArray(data)
42 out_ds.FlushCache()
```

← Only  
difference  
with prev  
example



- > MSc\_theses
- > PLUC\_UU
- ▼ python\_in\_GIS
  - > 2016\_2017
  - ▼ 2017\_2018
    - ▼ data
      - > GermanyDGM1
      - > naturalearthdata
      - ▼ rasters
        - test.tif



- ▼ ☒  **test**
  - 0
  - 1

IFGI



# Converting to other formats (1)

```
gdal3_change_format.py
1 import gdal
2 import numpy as np
3 import os
4 import osr
5
6 #####
7 data_dir = os.path.join('C:\\', 'Users', 'verstege', '\\
8 'Documents', 'education', 'python_in_GIS', '2017_2018', '\\
9 'data')
10 # Path to the old and new raster file
11 in_fn = os.path.join(data_dir, 'rasters', 'test.tif')
12 out_fn = os.path.join(data_dir, 'rasters', 'test.asc')
13
```

# Converting to other formats (2)

```
gdal3_change_format.py
10 # Path to the old and new raster file
11 in_fn = os.path.join(data_dir, 'rasters', 'test.tif')
12 out_fn = os.path.join(data_dir, 'rasters', 'test.asc')
13
14 driver = gdal.GetDriverByName('AAIGrid')
15 in_ds = gdal.Open(in_fn)
16 out_ds = driver.CreateCopy(out_fn, in_ds)
17 del out_ds
18
```

Some formats allow quick copies between data formats

# Converting to other formats (3)

In case direct conversion is not possible, you may have to:

- Create a new dataset in the format you want to have
- using the metadata (SRS, rows, cols etc.) from the original dataset
- Read the data from the original dataset as an array or as a byte sequence
- Copy the data into the new dataset

But dataset creation is not possible for all file formats, see:

[https://www.gdal.org/formats\\_list.html](https://www.gdal.org/formats_list.html)

# Selecting a subset of a raster (1)

```
gdal4_clip.py x
1 import gdal
2 import os
3
4 data_dir = os.path.join('C:\\', 'Users', 'verstege', '\\
5 'Documents', 'education', 'python_in_GIS', '2017_2018', '\\
6 'data')
7 # Path to the raster
8 in_fn = os.path.join(data_dir, 'rasters', 'test.tif')
9 out_fn = os.path.join(data_dir, 'rasters', '\\
10 | | | | | 'clipped_test.tif')
11 # Input: clip coordinates (max x and max y)
12 coordx = 10
13 coordy = 50
14
15 # Open the raster
16 rast_data_source = gdal.Open(in_fn)
```

← We write the clipped raster to this file

← We will clip the raster from the lower left corner up to this point

# Selecting a subset of a raster (2)

gdal4\_clip.py

```
17
18 # Get object that can translate coordinates
19 # to raster indices
20 # BEHAVIOR OF InvGeoTransform depends on gdal version
21 gt = rast_data_source.GetGeoTransform() ← This one goes from indices to coords
22 inv_gt = gdal.InvGeoTransform(gt) ← And this one the other way around
23
24 # Get the indices of the coordinates of your clip
25 x1, y1 = gdal.ApplyGeoTransform(inv_gt, coordx, coordy) ← Calculate index nrs
26 # make integers of these indices
27 x1 = int(x1)
28 y1 = int(y1)
29 print('column numbers are', x1, y1)
30 # for x nr of columns is the same as we clip from the x origin
31 out_columns = x1
32 # for y not
33 out_rows = rast_data_source.RasterYSize - y1
```

← If you do not clip from/to the border, you have to GeoTransform two points and subtract the indices




# Selecting a subset of a raster (3)

gdal4\_clip.py

```
35 # Create empty output raster (clipped size)
36 out_driver = gdal.GetDriverByName('GTiff')
37 # Rasters can be overwritten (cannot delete)
38 out_ds = out_driver.Create(out_fn, out_columns,
39                             out_rows, 1)
40 out_ds.SetProjection(rast_data_source.GetProjection())
41 # Geotransfor can remain the same, except the y origin!
42 out_gt = list(gt)
43 out_gt[3] = coordy
44 print(out_gt)
45 out_ds.SetGeoTransform(out_gt)
46
47 # Get data from the source raster and write to the new one
48 in_band = rast_data_source.GetRasterBand(1)
49 out_band = out_ds.GetRasterBand(1)
50 data = in_band.ReadAsArray(0, y1, out_columns, out_rows)
51 out_band.WriteArray(data)
52 out_ds.FlushCache()
53 print('done')
```



Change all parameters in the list that are different for the new data set (see also previous lecture)



From x,  
from y,  
cols, rows

# Result

\*Untitled Project - QGIS

Project Edit View Layer Settings Plugins Vector Raster Help

12.00000 meters

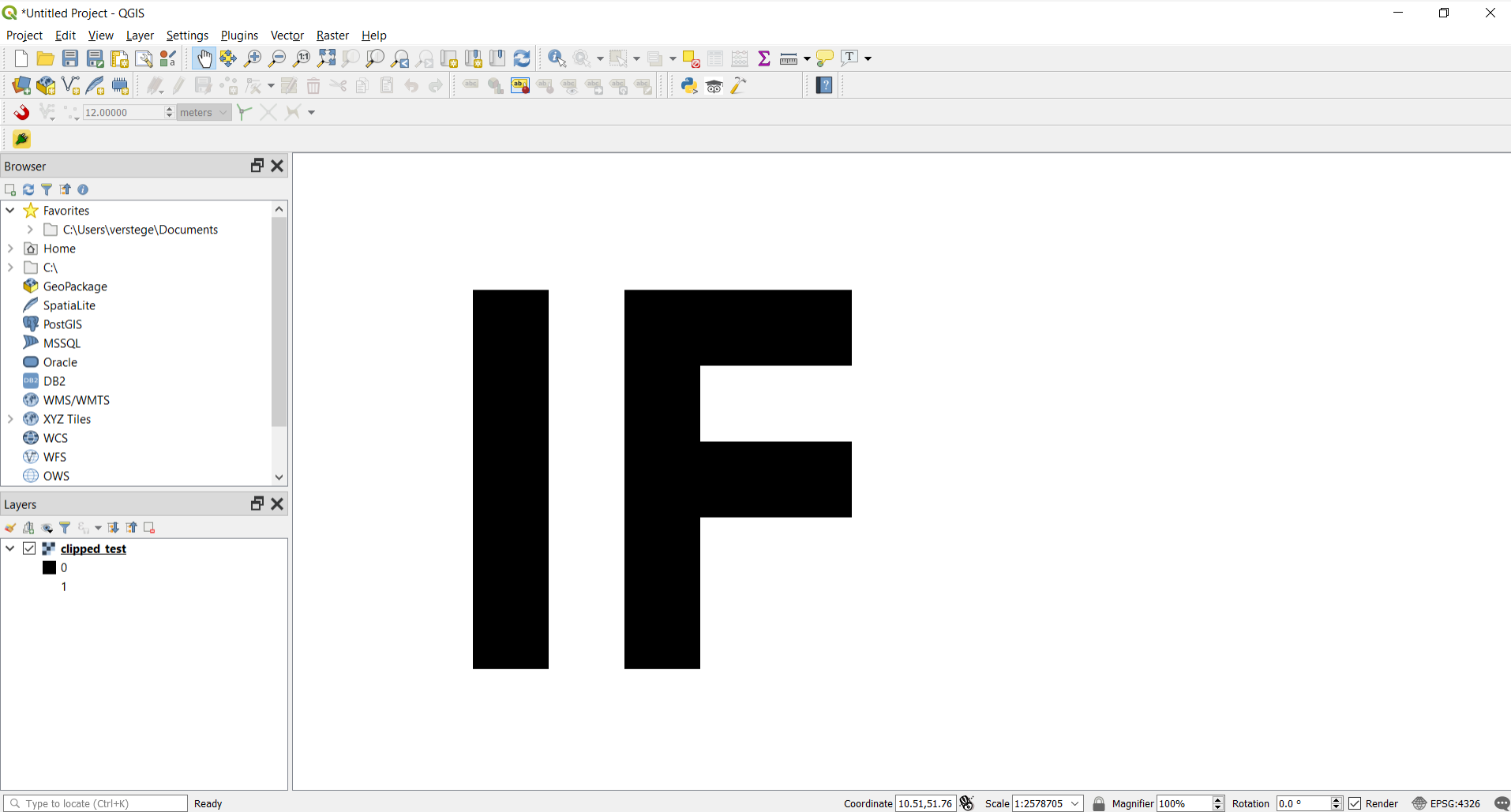
Browser

- ★ Favorites
  - C:\Users\verstege\Documents
- Home
- C:\
- GeoPackage
- Spatialite
- PostGIS
- MSSQL
- Oracle
- DB2
- WMS/WMTS
- XYZ Tiles
- WCS
- WFS
- OWS

Layers

- ☒ **clipped test**
  - 0
  - 1

Coordinate: 10.51,51.76 Scale: 1:2578705 Magnifier: 100% Rotation: 0.0° Render EPSG:4326



## Exercise #3

In the data folder there is a geotiff file `clipped_dem.tif`

Write a script, using ogr, to:

- open this file
- save the file as an asci grid

If you have time left:

- read the data from the geotiff file as an array
- print this array to see the data