WWU MÜNSTER

Python in GIS

OGR and GDAL

ifgi
Institut für Geoinformatik
Universität Münster

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

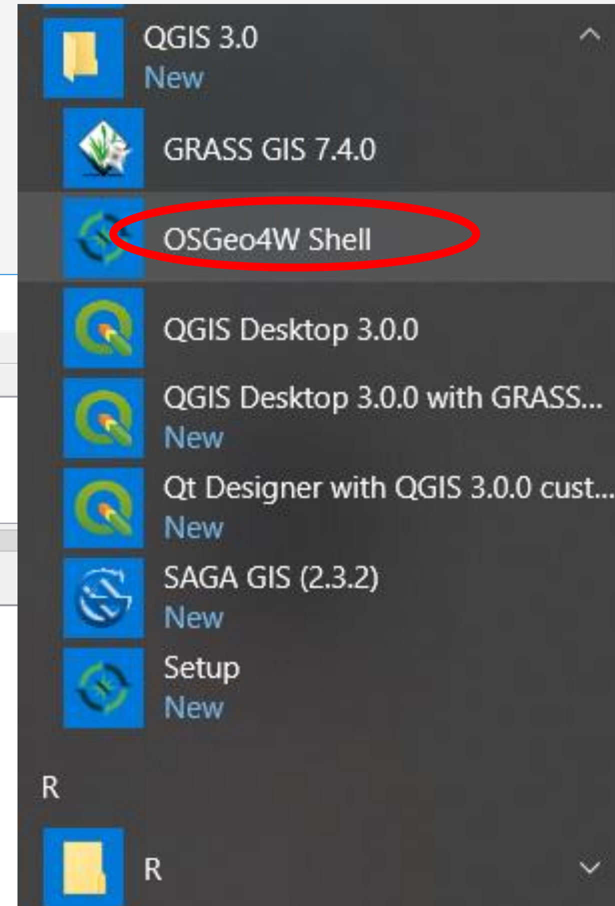
Acknowledgements:

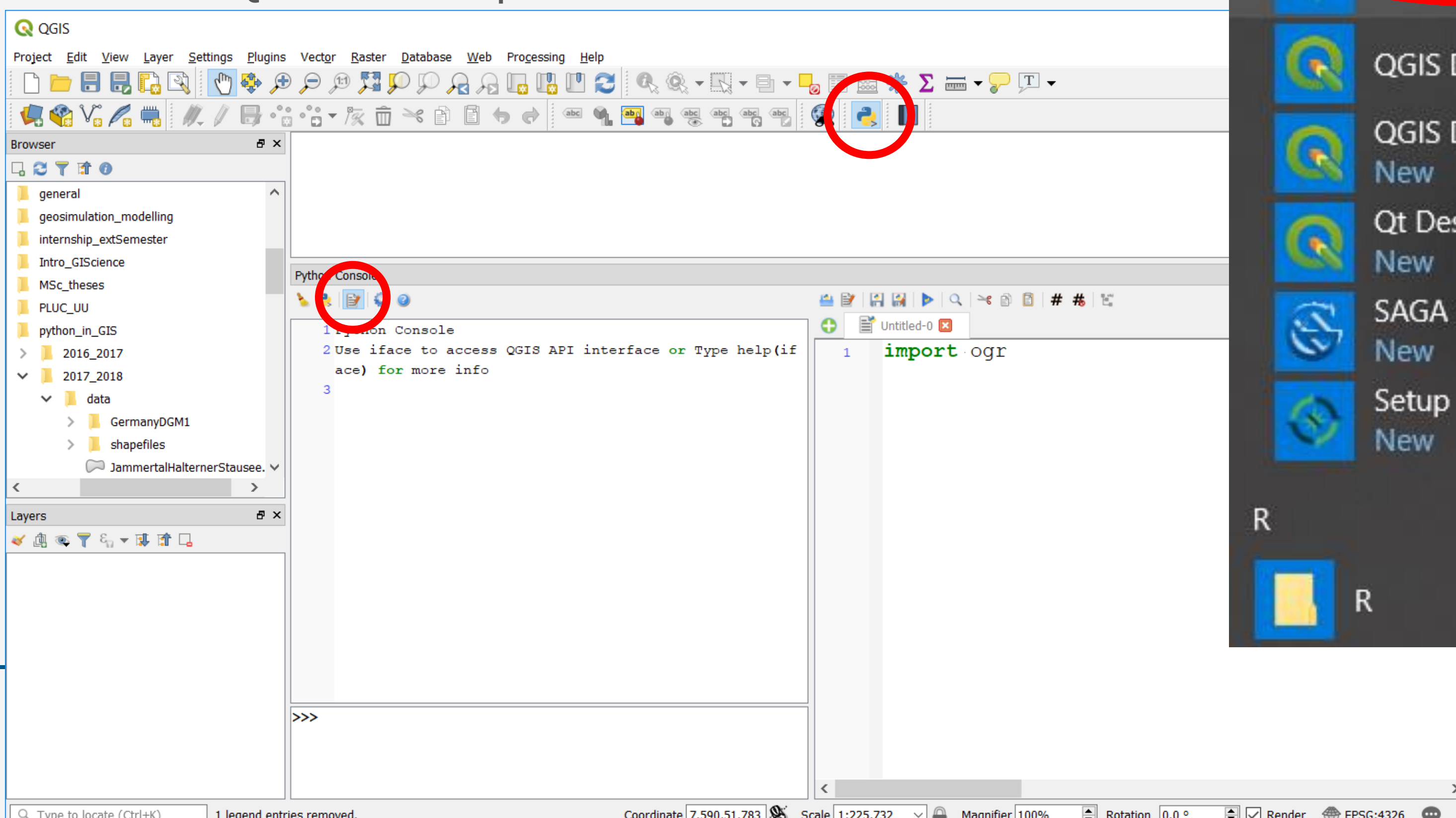
- The book Geoprocessing with Python by Chris Garrard
- The Python GDAL/OGR Cookbook: https://pcjericks.github.io/py-gdalogr-cookbook/index.html (Not yet adapted to QGIS 3)

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

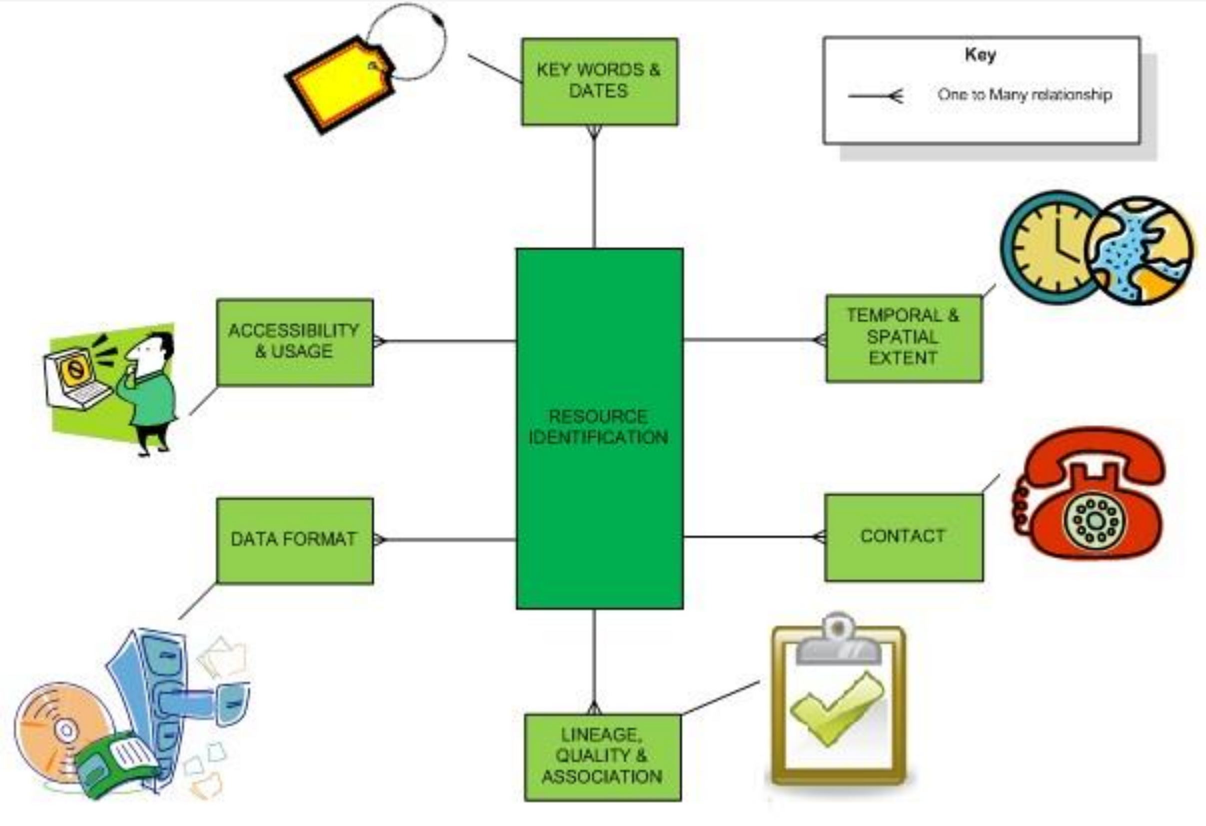
The Python API is not extremely well documented, but parallels the C/C++ APIs

* 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

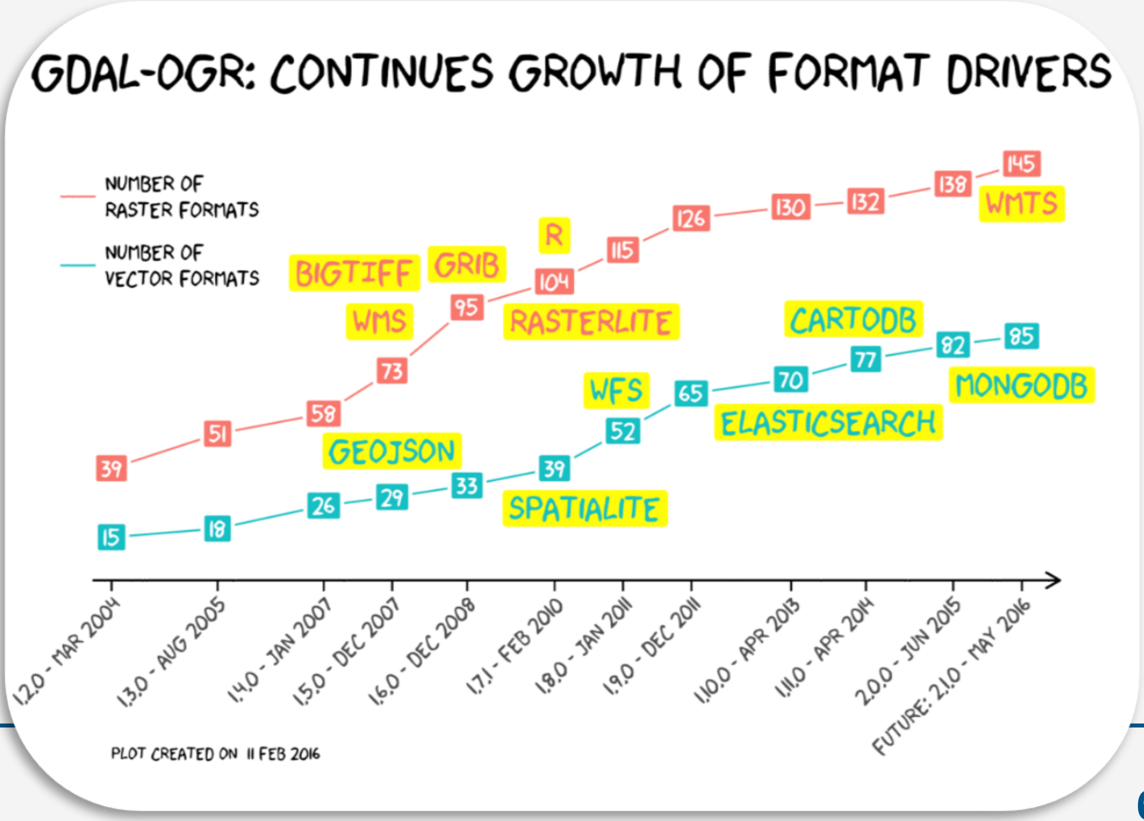
Drivers

You need to have an appropriate <u>driver</u> 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

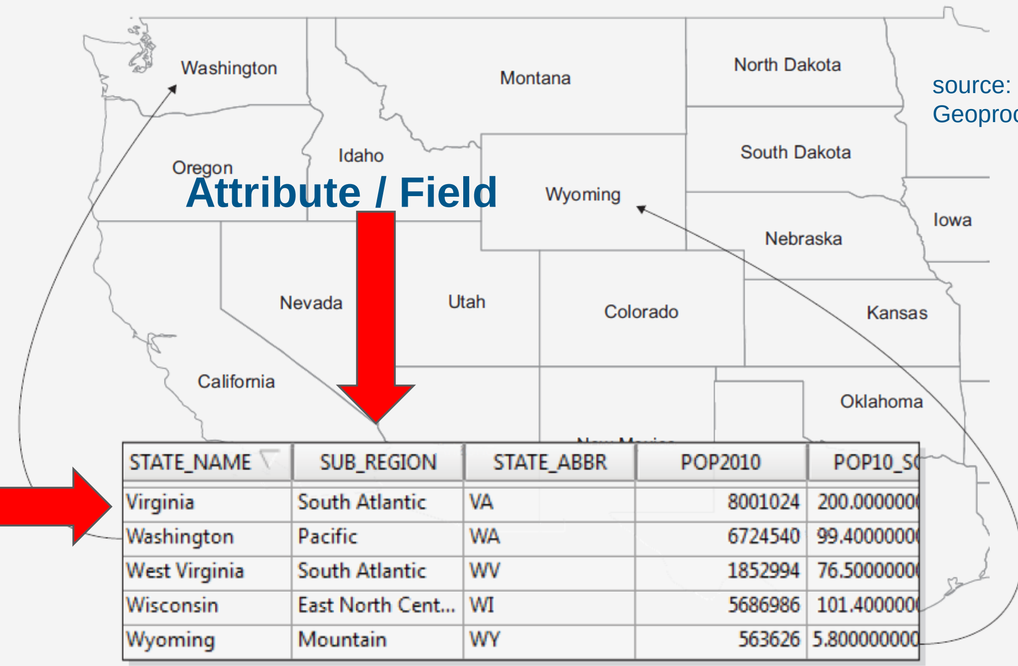
OGR

Vector data

source: Garrard, 2016, Geoprocessing with Python

Attribute / Field

Feature



STATE_NAME	SUB_REGION	STATE_ABBR	POP2010	POP10_S(
Virginia	South Atlantic	VA	8001024	200.000000(
Washington	Pacific	WA	6724540	99.4000000(
West Virginia	South Atlantic	WV	1852994	76.5000000(
Wisconsin	East North Cent...	WI	5686986	101.400000(
Wyoming	Mountain	WY	563626	5.800000000

Short recollection:

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

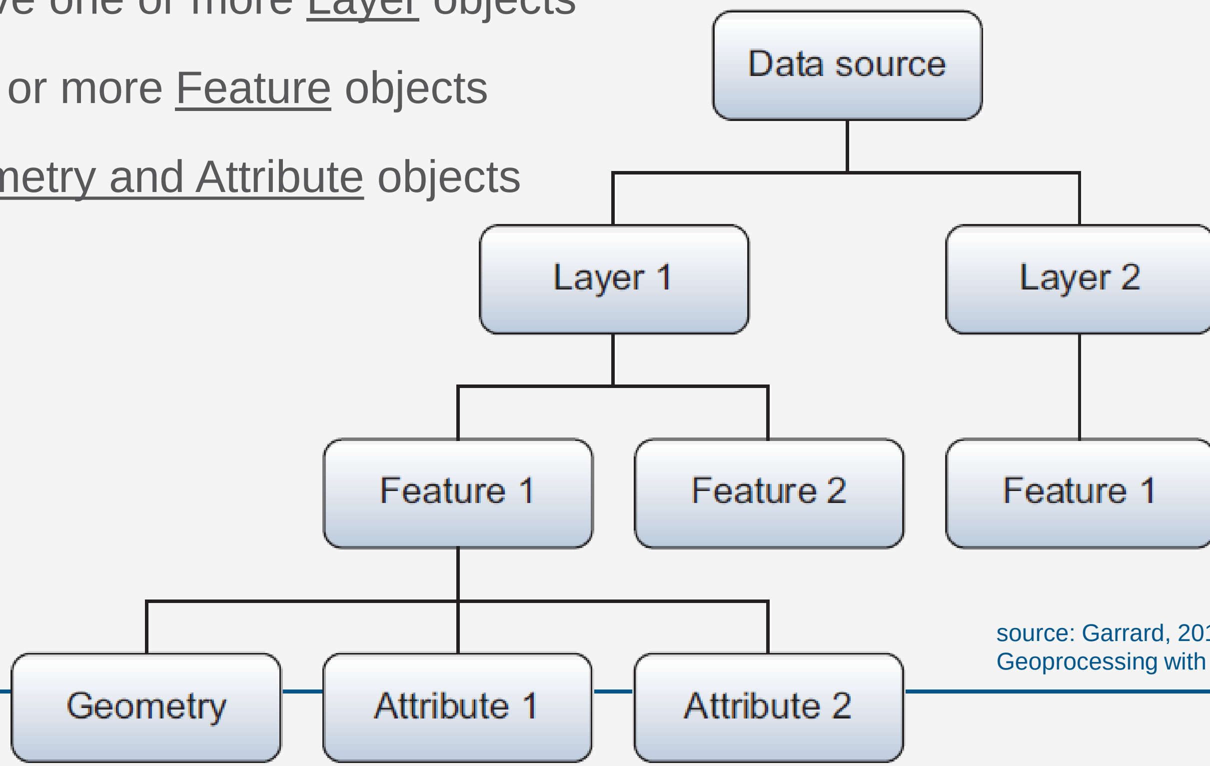
Geographic features with distinct boundaries, such as cities, work well as vector data, but continuous data, such as elevation, don’t.

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

WWU MÜNSTER

Checking available drivers in Python (1)

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

ogr1_drivers.py

```python
import ogr

#drivers

driver = ogr.GetDriver(1)
print(driver)
```

Python Console

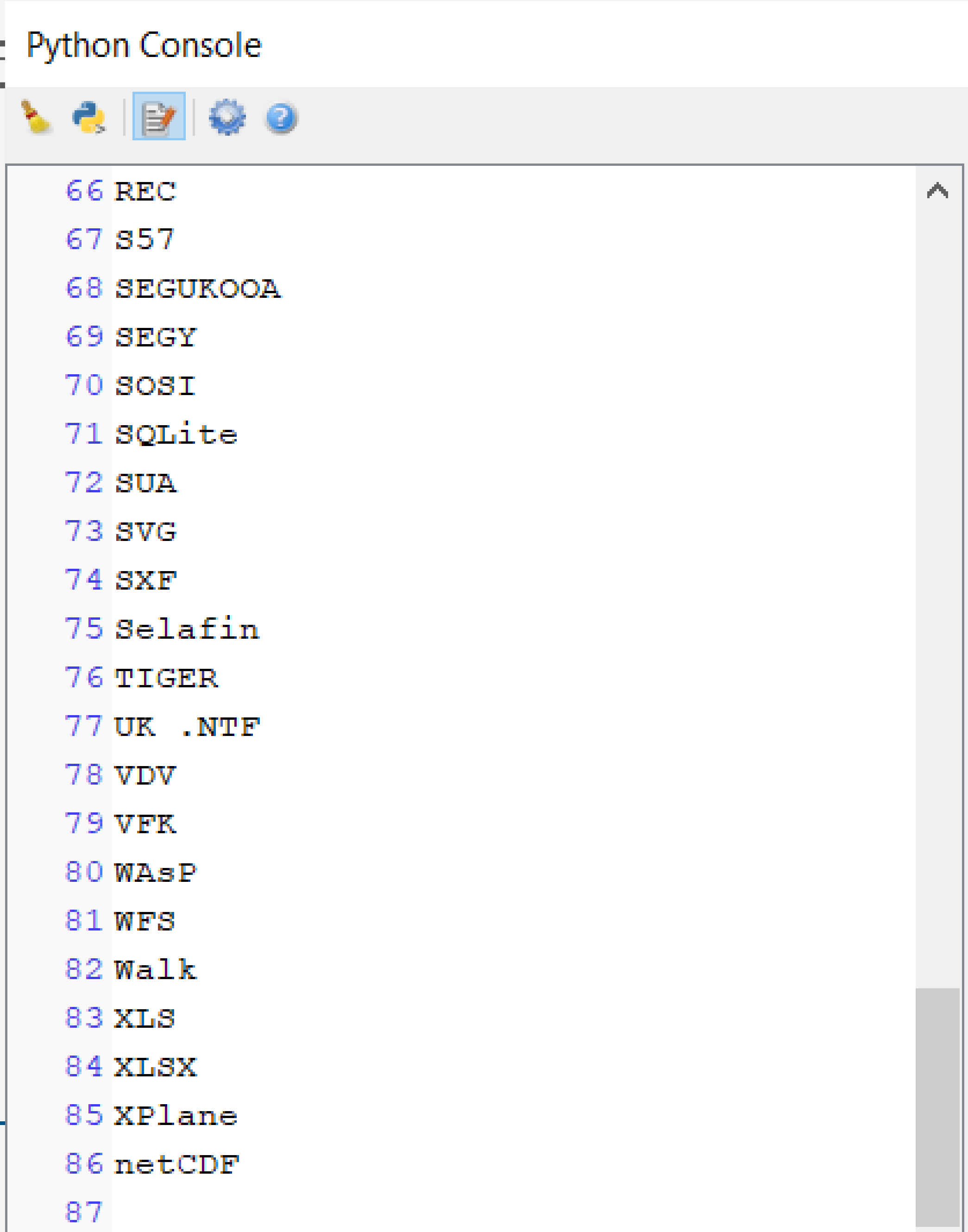
```
Python Console
Use iface to access QGIS API interface or Type help(iface) for more info
>>> exec(open('C:/Users/verstege/Documents/education/python_in_GIS/2017_2018/scripts/4_OGR1/ogr1_drivers.py'.encode('utf-8')).read())
<osgeo.ogr.Driver; proxy of <Swig Object of type 'OGRDriverShadow *' at 0x000001705DF69B70>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

Unless you want to see them all ...

Checking available drivers in Python (2)

ogr1_drivers.py

```
import ogr

#drivers
driver = ogr.GetDriver(1)
print(driver)

cnt = ogr.GetDriverCount()
formats_list = []   # Empty List

for i in range(cnt):
    driver = ogr.GetDriver(i)
    driver_name = driver.GetName()
    if not driver_name in formats_list:
        formats_list.append(driver_name)

formats_list.sort() # Sorting the list of drivers

for i in formats_list:
    print(i)
```

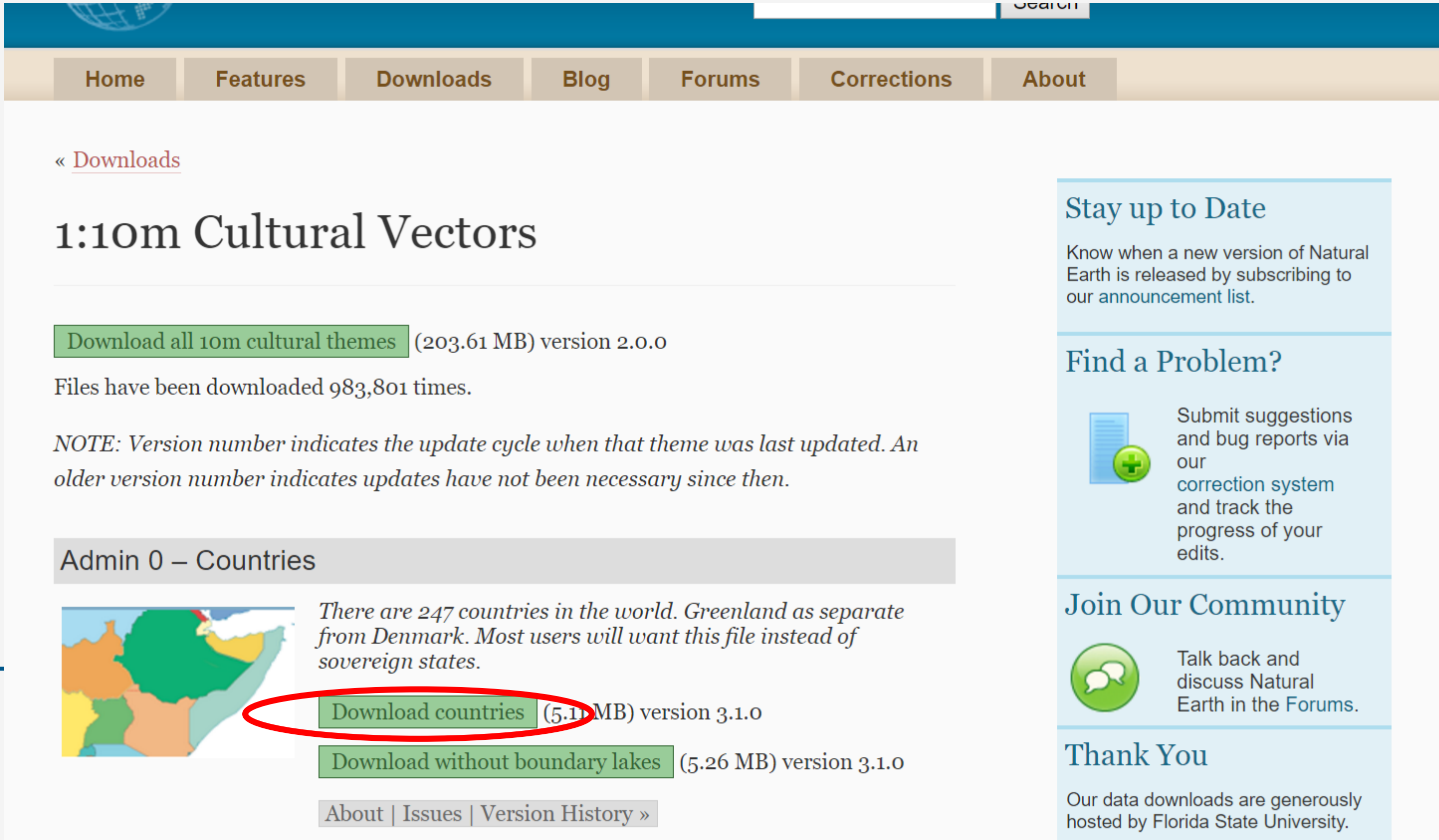

Python Console
66 REC
67 S57
68 SEGUKOOA
69 SEGY
70 SOSI
71 SQLite
72 SUA
73 SVG
74 SXF
75 Selafin
76 TIGER
77 UK .NTF
78 VDV
79 VFK
80 WAsP
81 WFS
82 Walk
83 XLS
84 XLSX
85 XPlane
86 netCDF
87

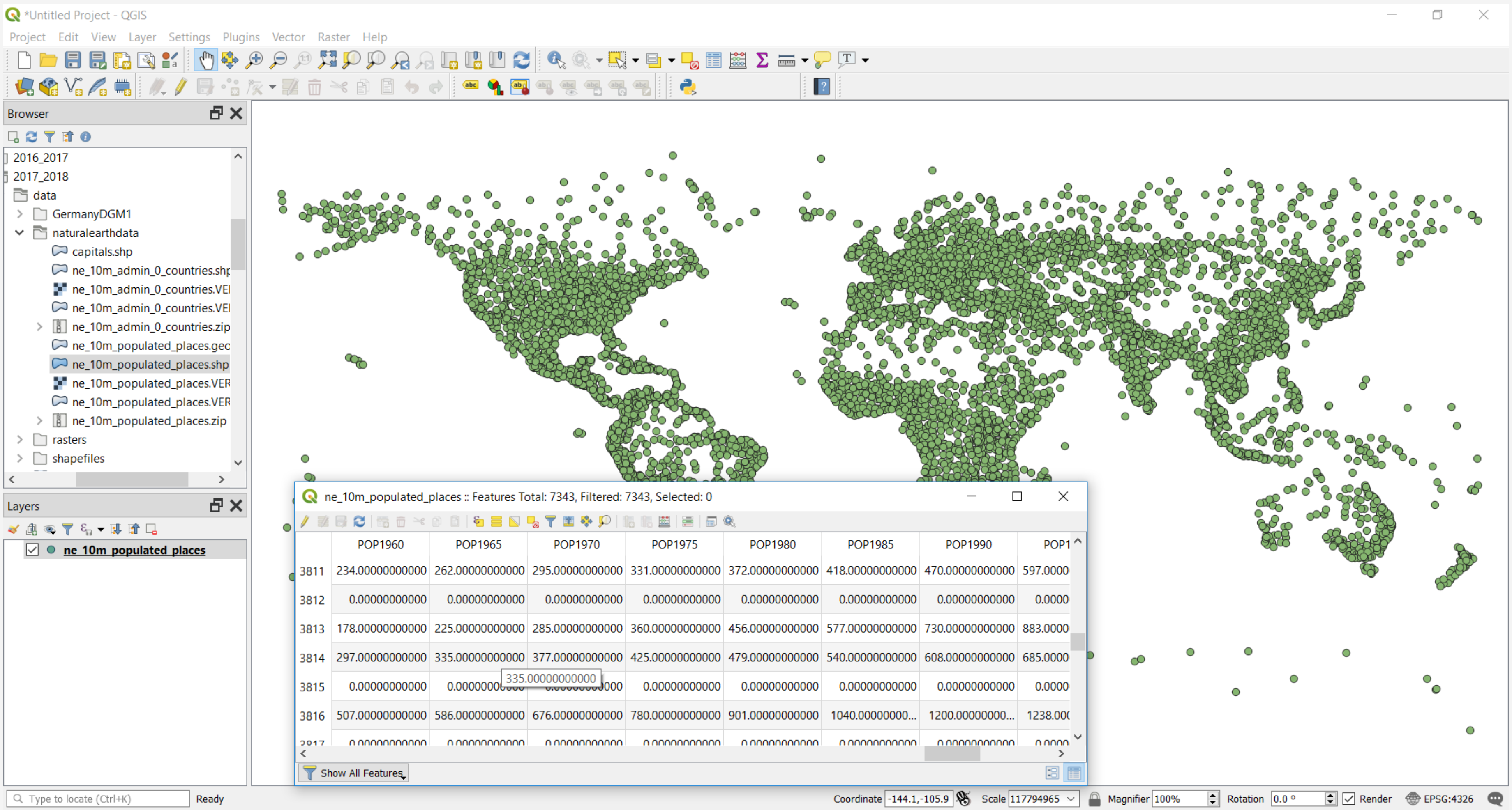
90 drivers!

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




*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.shp
ne_10m_admin_0_countries.VE
ne_10m_admin_0_countries.VE
ne_10m_admin_0_countries.zip
ne_10m_populated_places.gec
ne_10m_populated_places.shp
ne_10m_populated_places.VER
ne_10m_populated_places.VER
ne_10m_populated_places.zip
rasters
shapefiles
Layers
ne_10m_populated_places
ne_10m_populated_places :: Features Total: 7343, Filtered: 7343, Selected: 0
POP1960 POP1965 POP1970 POP1975 POP1980 POP1985 POP1990
3811 234.00000000000 262.00000000000 295.00000000000 331.00000000000 372.00000000000 418.00000000000 470.00000000000 597.0000
3812 0.00000000000 0.00000000000 0.00000000000 0.00000000000 0.00000000000 0.00000000000 0.00000000000 0.0000
3813 178.00000000000 225.00000000000 285.00000000000 360.00000000000 456.00000000000 577.00000000000 730.00000000000 883.0000
3814 297.00000000000 335.00000000000 377.00000000000 425.00000000000 479.00000000000 540.00000000000 608.00000000000 685.0000
3815 0.00000000000 0.00000000000 0.00000000000 0.00000000000 0.00000000000 0.00000000000 0.0000
335.00000000000
3816 507.00000000000 586.00000000000 676.00000000000 780.00000000000 901.00000000000 1040.00000000... 1200.00000000... 1238.00
Show All Features
Type to locate (Ctrl+K)
Ready
Coordinate -144.1,-105.9
Scale 117794965
Magnifier 100%
Rotation 0.0 °
Render
EPSG:4326

WWU MÜNSTER

Read a shapefile and access metadata (2)

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

ogr2_access.py

```
import ogr
import os

#############
# reading a shapefile
in_path = os.path.join('C:\\', 'Users', 'verstege', \
'Documents', 'education', 'python_in_GIS', '2017_2018', \
'data', 'naturalearthdata', 'ne_10m_populated_places.shp')

# get the correct driver
driver = ogr.GetDriverByName('ESRI Shapefile')

# 0 means read-only. 1 means writeable.
data_source = driver.Open(in_path, 0)
```

Why?

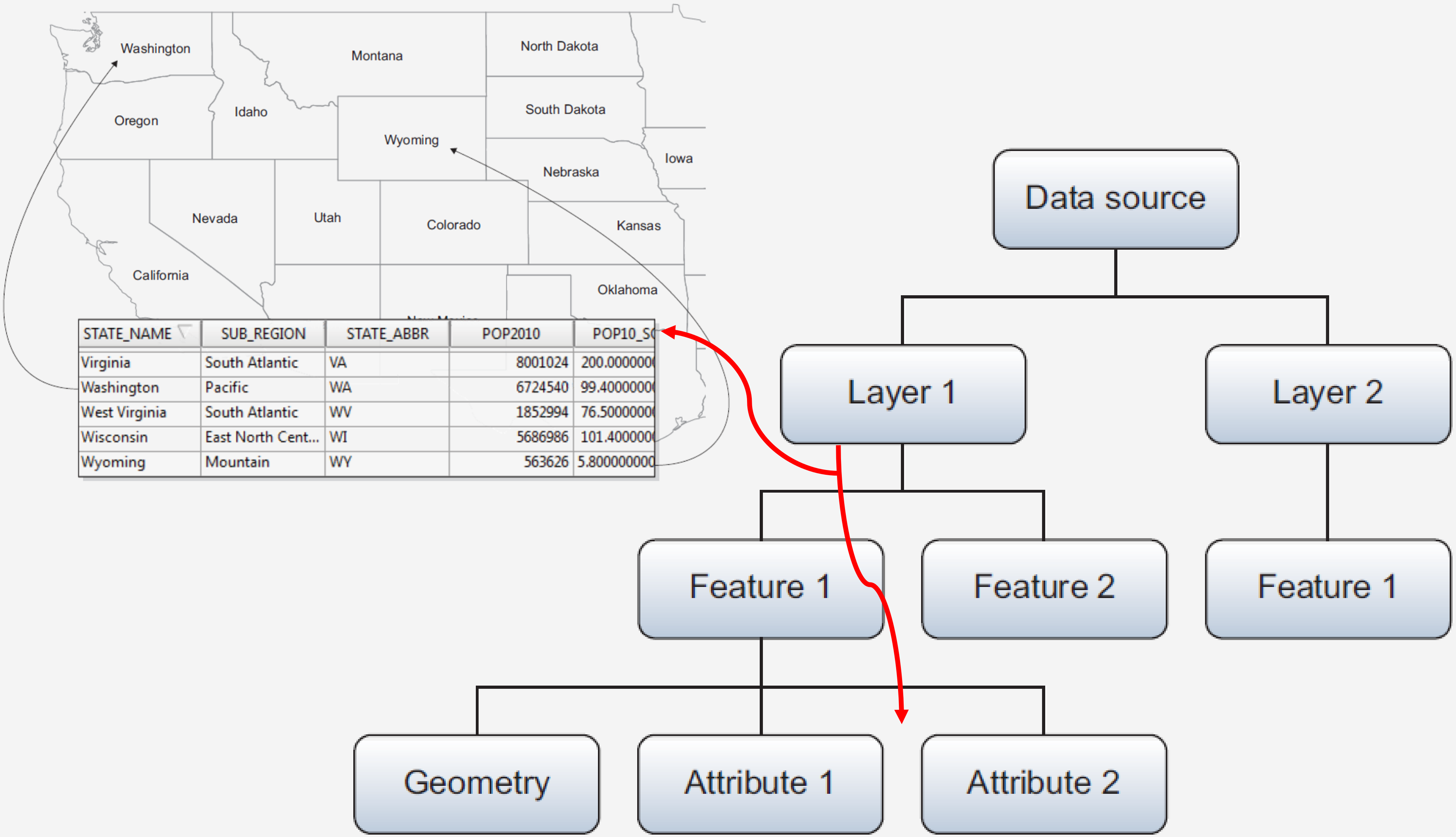
Read a shapefile and access metadata (3)

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

ogr2_access.py

```
# Check to see if shapefile is found.
if data_source is None:
    print('Could not open %s' % (in_path))

print('Opened %s' % (in_path))
# get the Layer class object
layer = data_source.GetLayer(0)
# get reference system info
spatial_ref = layer.GetSpatialRef()
print(spatial_ref)
# get info about the attributes (fields)
attributes = layer.GetLayerDefn()
for i in range(attributes.GetFieldCount()):
    print(attributes.GetFieldDefn(i).GetName())
```

Why?



STATE_NAME	SUB_REGION	STATE_ABBR	POP2010	POP10_S
Virginia	South Atlantic	VA	8001024	200.0000000
Washington	Pacific	WA	6724540	99.40000000
West Virginia	South Atlantic	WV	1852994	76.50000000
Wisconsin	East North Cent...	WI	5686986	101.4000000
Wyoming	Mountain	WY	563626	5.800000000

Read a shapefile and access metadata (4)

What is the format of the reference system info?

WKT - Well-Known Text

```
Python Console
Use iface to access QGIS API interface or Type help(iface) for more info
>>> exec(open('C:/Users/verstege/Documents/education/python_in_GIS/2017_2018/scripts/4_OGR1/ogr2_access.py'.encode('utf-8')).read())
Opened C:\Users\verstege\Documents\education\python_in_GIS\2017_2018\data\naturalearthdata\ne_10m_populated_places.shp
GEOGCS["GCS_WGS_1984",
    DATUM["WGS_1984",
        SPHEROID["WGS_84",6378
```

```
>>>
```

Python Console

```
POP1960
POP1965
POP1970
POP1975
POP1980
POP1985
POP1990
POP1995
POP2000
POP2005
POP2010
POP2015
POP2020
POP2025
POP2050
CITYALT
```

```
>>>
```

Read a shapefile and access metadata (5)

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

ogr2_access.py

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

Python Console

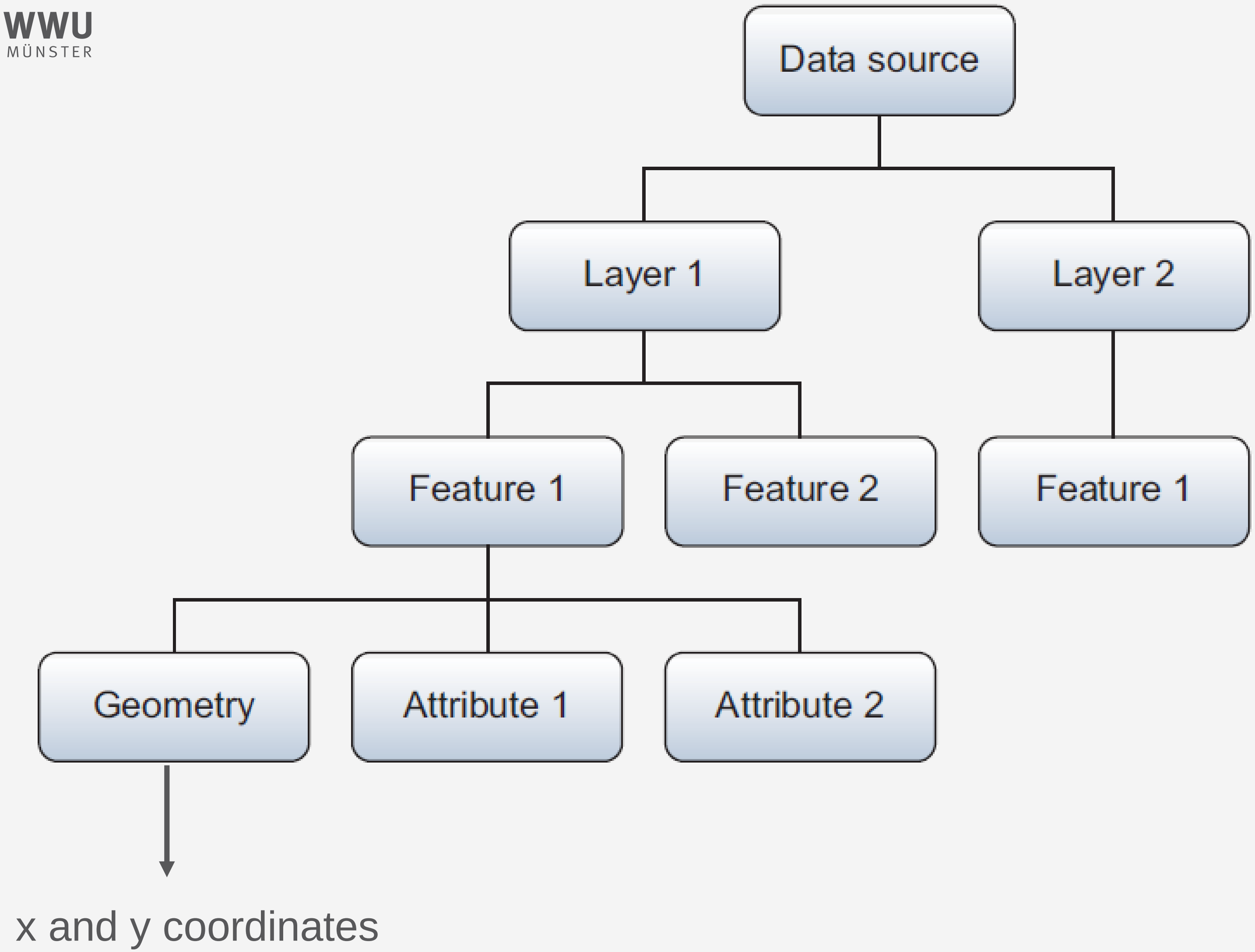
```
Number of features in ne_10m_populated_places.shp: 7343
```

Accessing features and attributes (1)

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

ogr2_access.py

```
# get info about the features
feature_count = layer.GetFeatureCount()
print("Number of features in %s: %d" % \
          (os.path.basename(in_path),feature_count))
# access single features
for feat in layer:
    pt = feat.geometry()
    x = pt.GetX()
    y = pt.GetY()
    name = feat.GetField('NAME')
    pop = feat.GetField('POP_MAX')
    country = feat.GetField('ADM0NAME')
    if country == 'Germany': print(name, pop, x, y)
```


Data source
Layer 1
Layer 2
Feature 1
Feature 2
Feature 1
Geometry
Attribute 1
Attribute 2
x and y coordinates

Accessing features and attributes (2)

Python Console

```
Number of features in ne_10m_populated_places.shp: 7343
Mainz 184997 8.27321915500556 49.9824724550127 8
Schwerin 96641 11.416698610531512 53.633304077430296
Bielefeld 331906 8.530011351158521 52.029988219300 04
Dortmund 588462 7.450025592690793 51.5299670604439 4
Duisburg 1276757 6.750016640865056 51.42997316395917 6
Wuppertal 776525 7.169991006100929 51.250009988502654
Essen 1742135 7.016615355058889 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.:

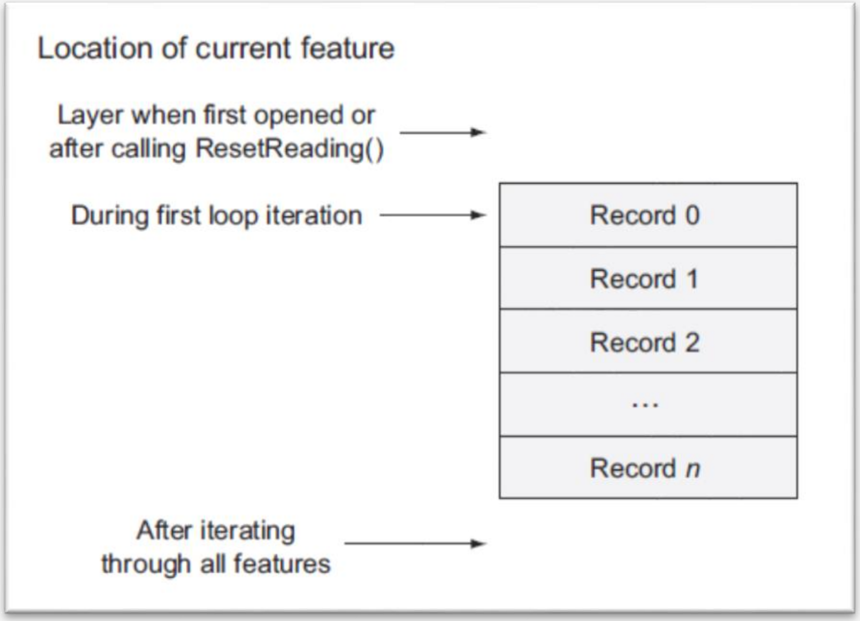
```
pop = feat.GetFieldAsString('POP_MAX')
```

```
# access single features
for feat in layer:
    pt = feat.geometry()
    x = pt.GetX()
    y = pt.GetY()
    name = feat.GetField('NAME')
    pop = feat.GetField('POP_MAX')
    country = feat.GetField('ADM0NAME')
    if country == 'Germany': print(name, pop, x, y)
```

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:

ogr3_attribute_filter.py

```python

print('Opened %s' % (in_path))
# get the Layer class object
layer = data_source.GetLayer(0)
# set a filter 
layer.SetAttributeFilter("ADM0NAME = 'Germany'")
# access single features
for feat in layer:
    pt = feat.geometry()
    x = pt.GetX()
    y = pt.GetY()
    name = feat.GetField('NAME')
    pop = feat.GetField('POP_MAX')
    print(name, pop, x, y)

```

Attribute filters (4)

Some more conditions you can use:

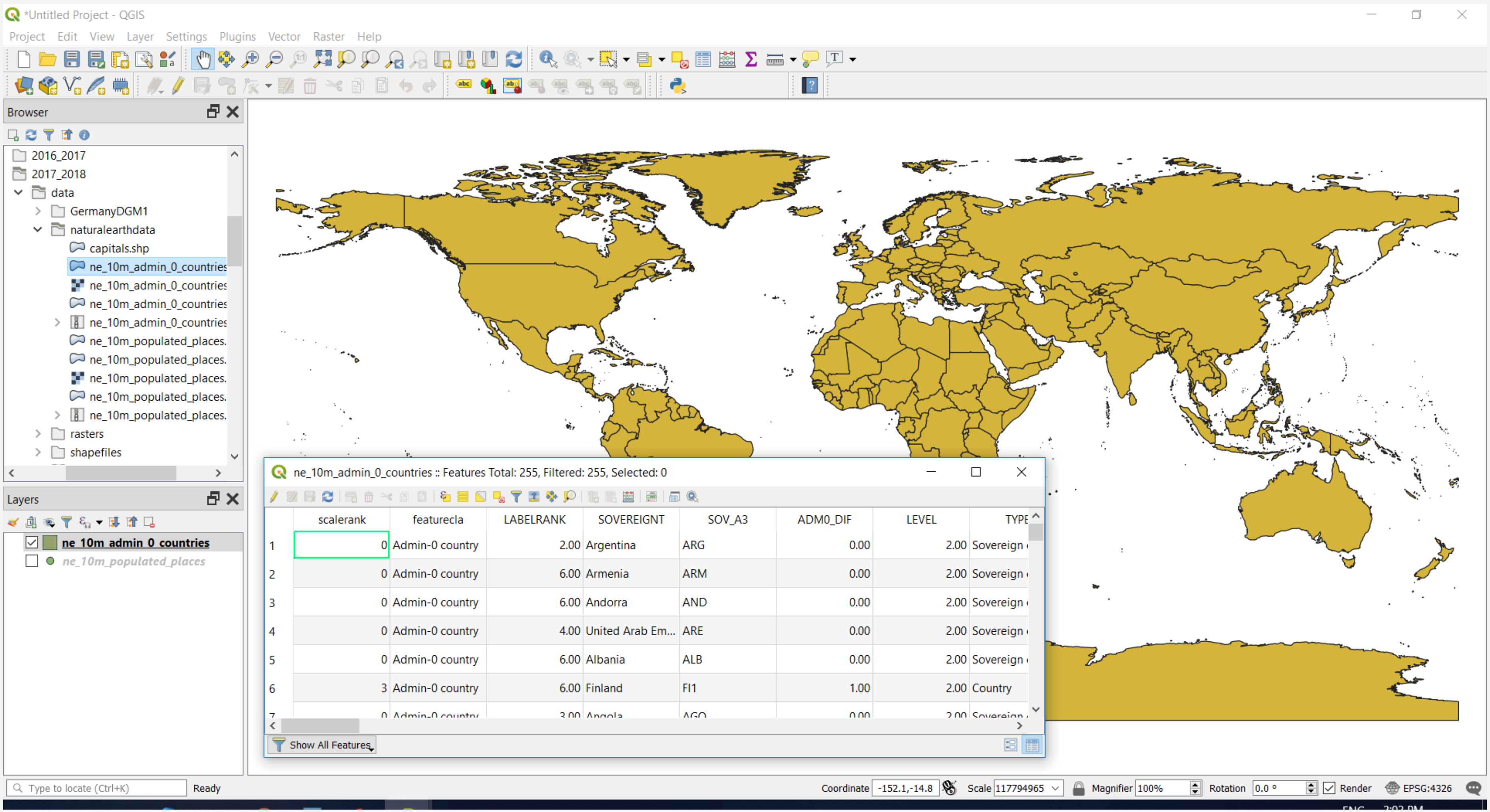
- Conditions can be negated using NOT, and NULL is used to indicate a null or no data value in the attribute table:
  - **'(Population < 50000) OR NOT (Place_type = "County Seat")'**
  - **'County NOT NULL‘**
- Ranges can be selected with BETWEEN or IN:
  - **'Population BETWEEN 25000 AND 50000'**
  - ***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


*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 FI1 1.00 2.00 Country
Show All Features
Type to locate (Ctrl+K)
Ready
Coordinate -152.1,-14.8
Scale 117794965
Magnifier 100%
Rotation 0.0 °
Render
EPSG:4326

WWU MÜNSTER

Spatial filters (2)

ogr4_spatial_filter.py

Copy the previous script, and now load two shapefiles

```
import ogr
import os

#############
# reading the places shapefile
in_path = os.path.join('C:\\', 'Users', 'verstege', \
'Documents', 'education', 'python_in_GIS', '2017_2018', \
'data', 'naturalearthdata', 'ne_10m_populated_places.shp')

# reading the countries shapefile
in_path2 = os.path.join('C:\\', 'Users', 'verstege', \
'Documents', 'education', 'python_in_GIS', '2017_2018', \
'data', 'naturalearthdata', 'ne_10m_admin_0_countries.shp')

# get the correct driver
driver = ogr.GetDriverByName('ESRI Shapefile')

# 0 means read-only. 1 means writeable.
places_source = driver.Open(in_path, 0)
countries_source = driver.Open(in_path2, 0)
```

Spatial filters (3)

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

ogr4_spatial_filter.py

```python

print('Opened %s' % (in_path2))
# get the Layer class objects
places_layer = places_source.GetLayer(0)
countries_layer = countries_source.GetLayer(0)
# set a filter to the countries layer
countries_layer.SetAttributeFilter("NAME = 'Germany'")
# get the feature (should be only one!)
feature_count = countries_layer.GetFeatureCount()
print("Number of features in %s: %d" % \
      (os.path.basename(in_path2),feature_count))
feat =  countries_layer.GetNextFeature()
germany = feat.geometry().Clone()
print(germany)
```

Attribute filter

Clone geometry

Python Console

```
Number of features in ne_10m_admin_0_countries.shp: 1
MULTIPOLYGON (((6.74219811300011 53.5783552100001,6.74952233200011 53.572414455,6.7565210300001 53.562892971,6.74756920700008 53.5659854190001,6.73462975400005 53.5751813820001,6.72608483200014 53.5771345070001,6.71599368600008 53.5760765650001,6.70118248800011 53.5714378930001,6.69507897200012 53.5702985700001,6.67709394600007 53.5756289730001,6.66334069100014 53.5873477230001,6.65951582100013 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

```python
print(germany)

# now filter the features in the places layer
places_layer.SetSpatialFilter(germany)
# access single features in the places layer
for feat in places_layer:
    pt = feat.geometry()
    x = pt.GetX()
    y = pt.GetY()
    name = feat.GetField('NAME')
    print(name, x, y)

```

[Spatial filter](#)

Spatial filters (5)

Result

Python Console

```
Cologne 6.948058575221637 50.93194954014825
Dresden 13.750002806257669 51.04997051910084
Frankfurt 8.675015420169643 50.09997682606331
Hamburg 9.998053285520314 53.55197049556244
Munich 11.573047588912061 48.131887894629244
Berlin 13.399602764700546 52.523764522251156

>>>
```

Writing vector data to a different file format

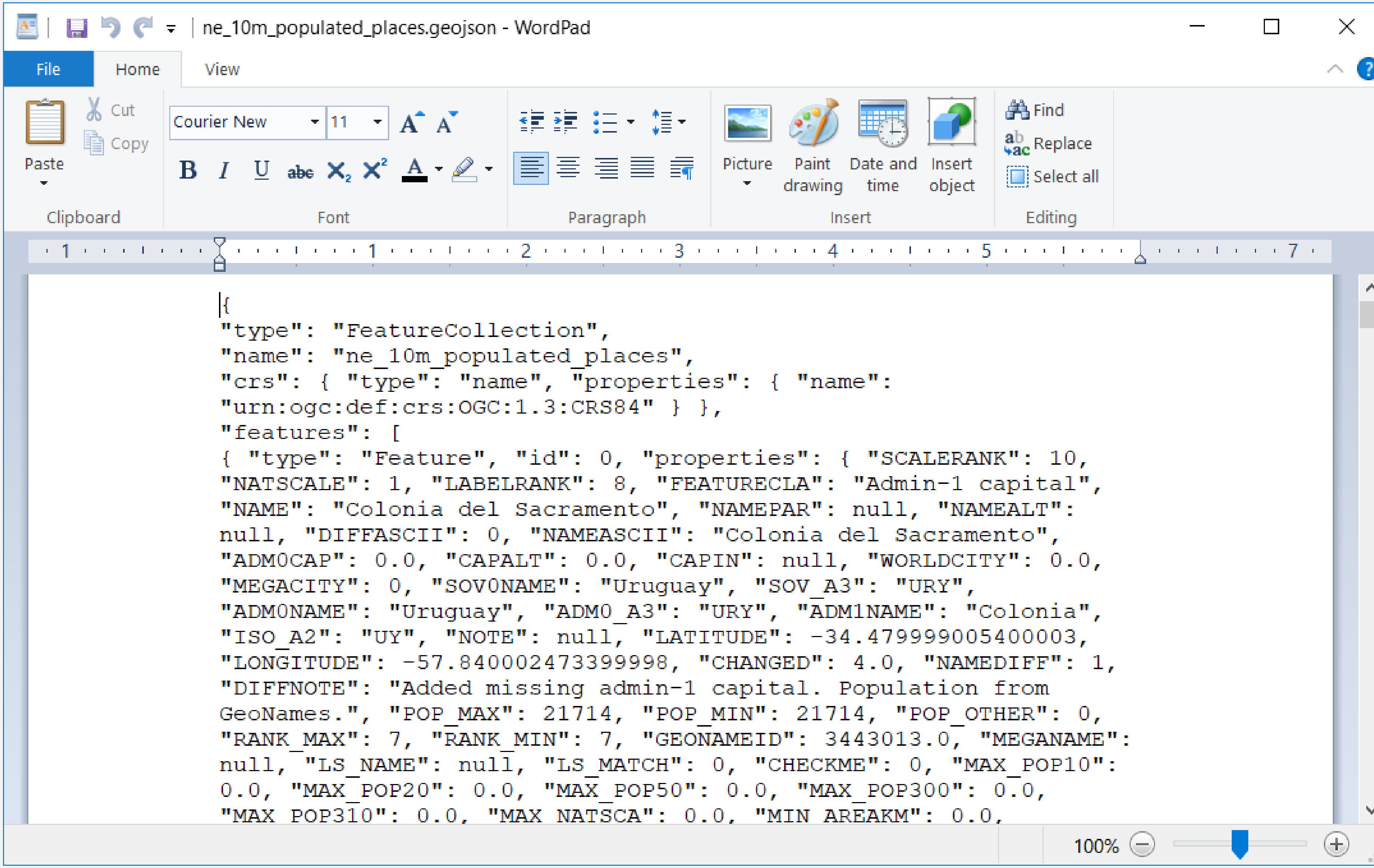
ogr5_write.py

To save all data in file to a new format: `CopyDataSource()`.

```
import ogr
import os

# reading a shapefile
in_path = os.path.join('C:\\', 'Users', 'verstege', \
'Documents', 'education', 'python_in_GIS', '2017_2018', \
'data', 'naturalearthdata', 'ne_10m_populated_places.shp')

# get the correct driver
in_driver = ogr.GetDriverByName('ESRI Shapefile')
places_source = in_driver.Open(in_path, 0)

# write the data to a geojson
out_file = os.path.join('C:\\', 'Users', 'verstege', \
'Documents', 'education', 'python_in_GIS', '2017_2018', \
'data', 'naturalearthdata', 'ne_10m_populated_places.geojson')
out_driver = ogr.GetDriverByName('GeoJSON')
out_ds  = out_driver.CopyDataSource(places_source,out_file)
del out_ds
```

Choose output type

To release the file



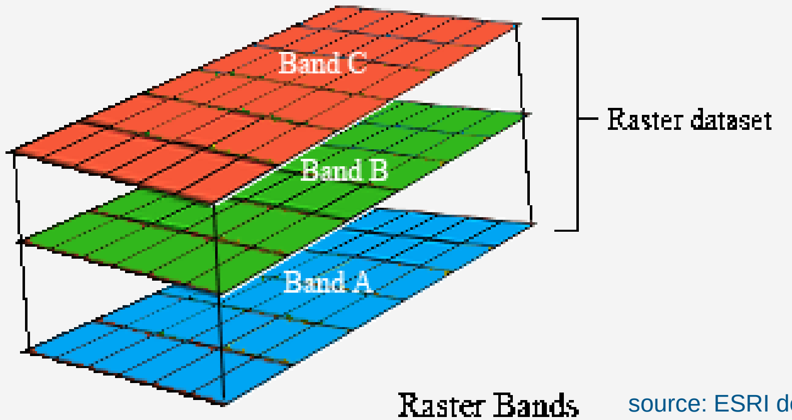
{
"type": "FeatureCollection",
"name": "ne_10m_populated_places",
"crs": { "type": "name", "properties": { "name":
"urn:ogc:def:crs:OGC:1.3:CRS84" } },
"features": [
{ "type": "Feature", "id": 0, "properties": { "SCALERANK": 10,
"NATSCALE": 1, "LABELRANK": 8, "FEATURECLA": "Admin-1 capital",
"NAME": "Colonia del Sacramento", "NAMEPAR": null, "NAMEALT":
null, "DIFFASCII": 0, "NAMEASCII": "Colonia del Sacramento",
"ADM0CAP": 0.0, "CAPALT": 0.0, "CAPIN": null, "WORLDCITY": 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,

Exercise #2

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

GDAL

Raster data



source: ESRI developer help

Short recollection:

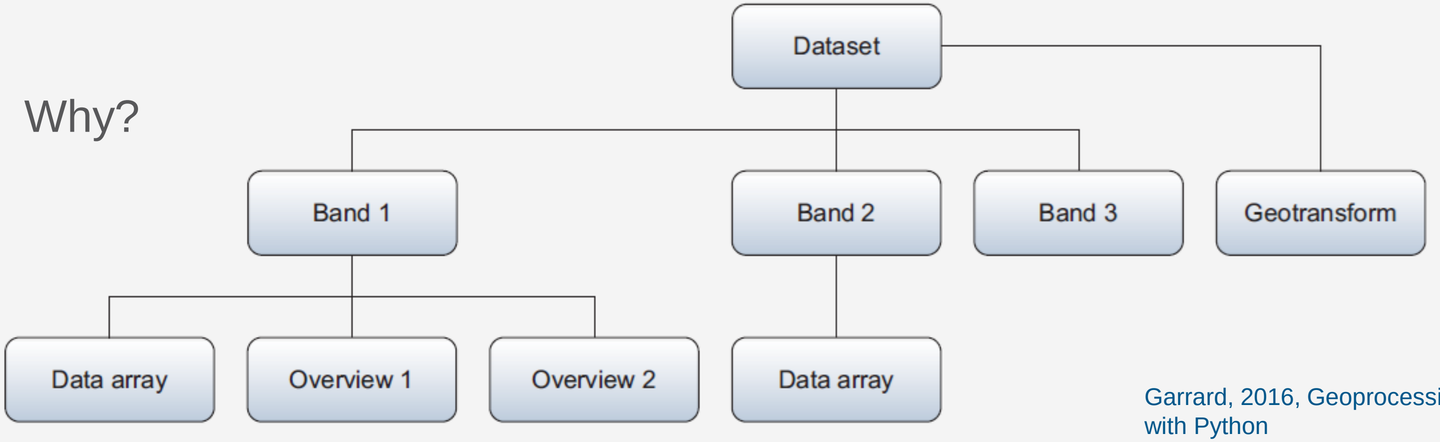
- In *raster data* geographic features are represented as <u>surfaces</u>, divided into a grid of equally sized cells.
- Geographic features that are <u>continuous</u>, such as elevation, work well as raster data, but discrete objects, such as houses, do not.
- Each band in raster data represents a 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

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

Why?



Garrard, 2016, Geoprocessing with Python

Reading a raster (1)

gdal0_read_raster.py

```
import gdal
import numpy as np
import os

#############
data_dir = os.path.join('C:\\', 'Users', 'verstege', \
'Documents', 'education', 'python_in_GIS', 'data')

# Path to the raster
in_path = os.path.join(data_dir, 'rasters', \
    'clipped_dem.tif')

# Open the raster
rast_data_source = gdal.Open(in_path)
```

Import the gdal library

Create path to a raster

Open without specifying the driver

WWU MÜNSTER

Reading a raster (2)

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

gdal0_read_raster.py

```
# Get metadata at data_source level
print('Nr of bands:', rast_data_source.RasterCount)
cols = rast_data_source.RasterXSize
rows = rast_data_source.RasterYSize
print('Size:', cols, rows)

# Select (the only) band and get meta data at band level
srcband = rast_data_source.GetRasterBand(1)
srcband.ComputeStatistics(0)
print('Minimum is:', srcband.GetMinimum())
print('Maximum is:', srcband.GetMaximum())

# Create empty array and catch the data in it
data = np.empty((rows, cols))
srcband.ReadAsArray(buf_obj=data)
print(data)
```

Get meta data

Band indices start at 1!!

Reading data with buffer

Reading a raster (3)

Result

Python Console

```
Use iface to access QGIS API interface or Type help(iface) for more info
Security warning: typing commands from an untrusted source can lead to data loss and/or leak
>>> exec(open('C:/Users/verstege/Documents/workshops_conferences/2019_2020/ILS_Python/materials/3_OGR_GDAL/gdal0_read_raster.py'.encode('utf-8')).read())
Nr of bands: 1
Size: 2463 1809
Minimum is: 32.0
Maximum is: 139.0
[[65. 64. 65. ... 64. 64. 64.]
 [64. 64. 65. ... 64. 64. 64.]
 [64. 65. 65. ... 64. 64. 64.]
 ...
 [73. 73. 73. ... 47. 47. 47.]
 [73. 73. 73. ... 47. 47. 47.]
 [73. 73. 73. ... 47. 47. 47.]]
```

WWU
MÜNSTER

Creating a new raster and adding data (1)

gdal1_new_raster.py

```
import gdal
import numpy as np
import os
import osr

#############
data_dir = os.path.join('C:\\', 'Users', 'verstege', \
'Documents', 'education', 'python_in_GIS', '2017_2018', \
'data')
# Path to the new raster file
out_fn = os.path.join(data_dir, 'rasters', 'test.tif')
# Spatial reference system
srs = osr.SpatialReference()
srs.ImportFromEPSG(4326) # WGS84
# Create the data
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

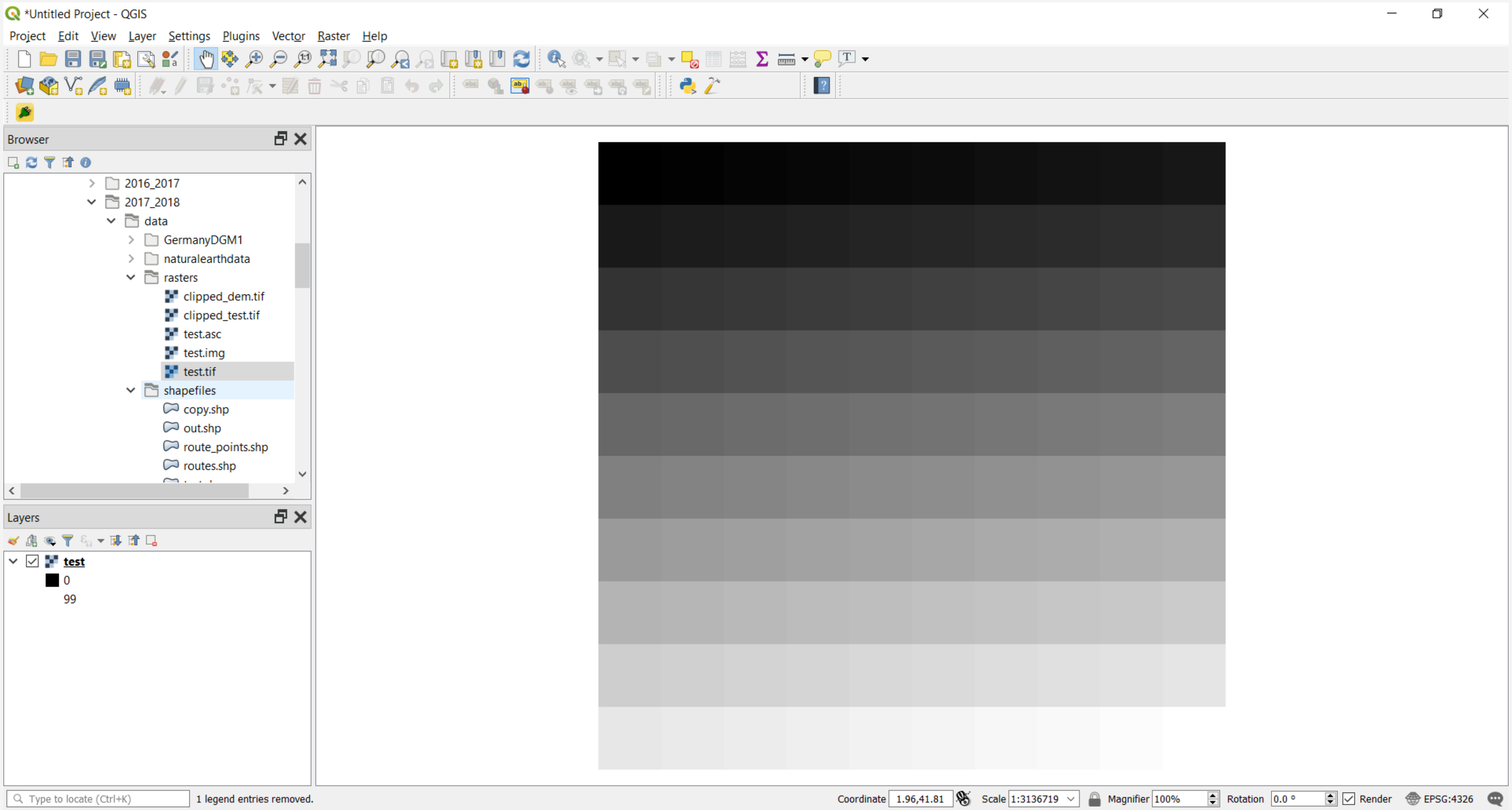
```
# Define properties of the new raster
originX = 4
originY = 52
pixelWidth = 1
pixelHeight = -1

# Create the output file and add the properties
driver = gdal.GetDriverByName('GTiff')
out_ds = driver.Create(out_fn, data.shape[1], 
                        data.shape[0], 1, gdal.GDT_UInt32)
out_ds.SetGeoTransform((originX, pixelWidth, 0, \
                        originY, 0, pixelHeight))
out_ds.SetProjection(srs.ExportToWkt())

# Add the data
outband = out_ds.GetRasterBand(1)
outband.WriteArray(data)
out_ds.FlushCache()

```

columns, rows, bands, data type

spatial reference info as a tuple

add the NumPy array to the band

Result


*Untitled Project - QGIS
Project Edit View Layer Settings Plugins Vector Raster Help
Browser
2016_2017
2017_2018
data
GermanyDGM1
naturalearthdata
rasters
clipped_dem.tif
clipped_test.tif
test.asc
test.img
test.tif
shapefiles
copy.shp
out.shp
route_points.shp
routes.shp
Layers
test
0
99
Type to locate (Ctrl+K)
1 legend entries removed.
Coordinate 1.96,41.81
Scale 1:3136719
Magnifier 100%
Rotation 0.0 °
Render
EPSG:4326

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

```python
# Srs
srs = osr.SpatialReference()
srs.ImportFromEPSG(4326) # WGS84
# Create the data
data = np.array([[1,1,1,1,1,1,1,1,1,1,1,1,1],
                 [1,1,1,1,1,1,1,1,1,1,1,1,1],
                 [1,0,1,0,0,0,1,0,0,0,1,0,1],
                 [1,0,1,0,1,1,1,0,1,1,1,0,1],
                 [1,0,1,0,0,0,1,0,1,0,1,0,1],
                 [1,0,1,0,1,1,1,0,1,0,1,0,1],
                 [1,0,1,0,1,1,1,0,0,0,1,0,1],
                 [1,1,1,1,1,1,1,1,1,1,1,1,1],
                 [1,1,1,1,1,1,1,1,1,1,1,1,1],
                 [1,1,1,1,1,1,1,1,1,1,1,1,1]])
originX = 4
originY = 52
pixelWidth = 1
pixelHeight = -1
```

a binary array

Again, cell size in y-direction always negative

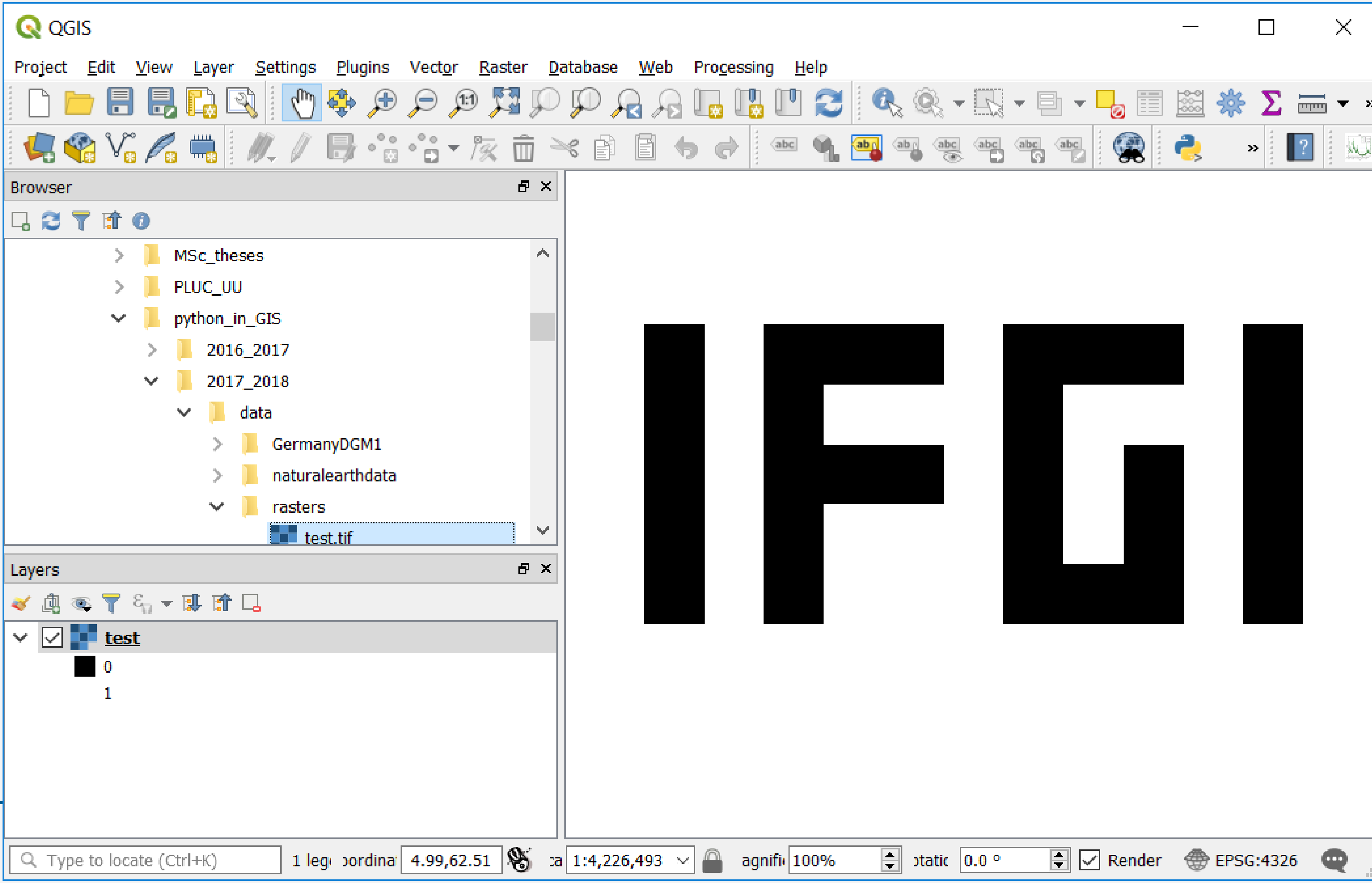
Other data type, example (2)

gdal2_edit_data.py

```
# Create the output file and set parameters
driver = gdal.GetDriverByName('GTiff')
out_ds = driver.Create(out_fn, data.shape[1],
                       data.shape[0], 1, gdal.GDT_Byte)
out_ds.SetGeoTransform((originX, pixelWidth, 0, \
                        originY, 0, pixelHeight))
out_ds.SetProjection(srs.ExportToWkt())

outband = out_ds.GetRasterBand(1)
outband.WriteArray(data)
out_ds.FlushCache()
```

Only difference with prev example

WWU MÜNSTER


QGIS
Project Edit View Layer Settings Plugins Vector Raster Database Web Processing Help
Browser
MSc_theses
PLUC_UU
python_in_GIS
2016_2017
2017_2018
data
GermanyDGM1
naturalearthdata
rasters
test.tif
Layers
test
0
1
IFGI
Type to locate (Ctrl+K)
4.99,62.51
1:4,226,493
100%
0.0 °
Render
EPSG:4326

WWU MÜNSTER

Converting to other formats (1)

gdal3_change_format.py

```
import gdal
import numpy as np
import os
import osr

#############
data_dir = os.path.join('C:\\', 'Users', 'verstege', \
'Documents', 'education', 'python_in_GIS', '2017_2018', \
'data')
# Path to the old and new raster file
in_fn = os.path.join(data_dir, 'rasters', 'test.tif')
out_fn = os.path.join(data_dir, 'rasters', 'test.asc')
```

WWU MÜNSTER

Converting to other formats (2)

gdal3_change_format.py

```
# Path to the old and new raster file
in_fn = os.path.join(data_dir, 'rasters', 'test.tif')
out_fn = os.path.join(data_dir, 'rasters', 'test.asc')

driver = gdal.GetDriverByName('AAIGrid')
in_ds = gdal.Open(in_fn)
out_ds = driver.CreateCopy(out_fn, in_ds)
del out_ds
```

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

Selecting a subset of a raster (1)

gdal4_clip.py

```
import gdal
import os

data_dir = os.path.join('C:\\', 'Users', 'verstege', \
'Documents', 'education', 'python_in_GIS', '2017_2018', \
'data')
# Path to the raster
in_fn = os.path.join(data_dir, 'rasters', 'test.tif')
out_fn = os.path.join(data_dir, 'rasters', \
                        'clipped_test.tif')
# Input: clip coordinates (max x and max y)
coordx = 10
coordy = 50

# Open the raster
rast_data_source = gdal.Open(in_fn)
```

We write the clipped raster to this file

We will clip the raster from the <u>lower left corner</u> up to this point

Selecting a subset of a raster (2)

gdal4_clip.py

```
# Get object that can translate coordinates 
# to raster indices
# BEHAVIOR OF InvGeoTransform depends on gdal version
gt = rast_data_source.GetGeoTransform()
inv_gt = gdal.InvGeoTransform(gt)

# Get the indices of the coordinates of your clip
x1, y1= gdal.ApplyGeoTransform(inv_gt, coordx, coordy)
# make integers of these indices
x1 = int(x1)
y1 = int(y1)
print('column numbers are', x1, y1)
# for x nr of columns is the same as we clip from the x origin
out_columns = x1
# for y not
out_rows = rast_data_source.RasterYSize - y1
```

This one goes from indices to coords

And this one the other way around

Calculate index nrs

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)

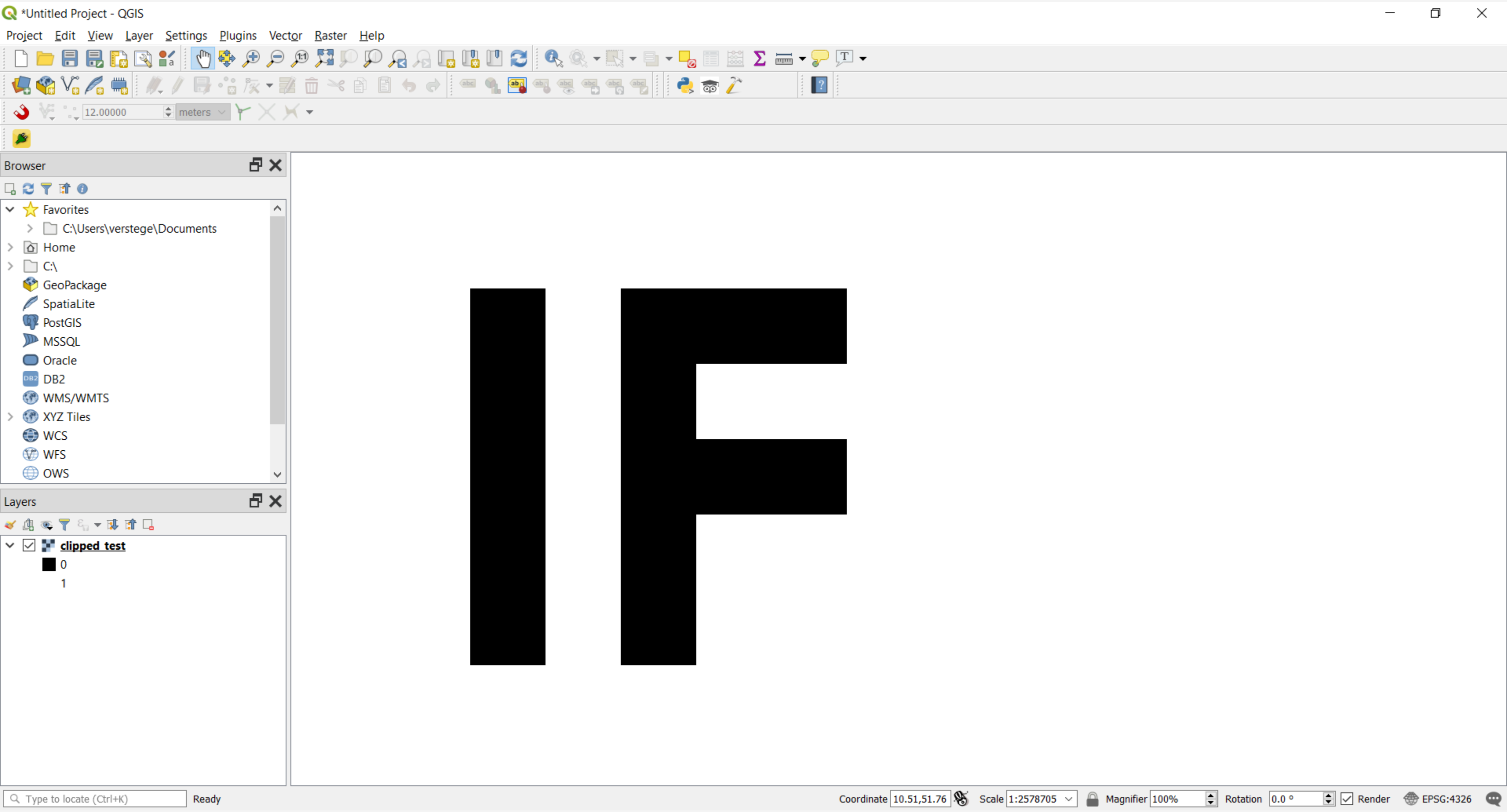
gdal4_clip.py

```
# Create empty output raster (clipped size)
out_driver = gdal.GetDriverByName('GTiff')
# Rasters can be overwritten (cannot delete)
out_ds = out_driver.Create(out_fn, out_columns,
                              out_rows, 1)
out_ds.SetProjection(rast_data_source.GetProjection())
# Geotransfor can remain the same, except the y origin!
out_gt = list(gt)
out_gt[3] = coordy
print(out_gt)
out_ds.SetGeoTransform(out_gt)

# Get data from the source raster and write to the new one
in_band = rast_data_source.GetRasterBand(1)
out_band = out_ds.GetRasterBand(1)
data = in_band.ReadAsArray(0, y1, out_columns, out_rows)
out_band.WriteArray(data)
out_ds.FlushCache()
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
Type to locate (Ctrl+K)
Ready
Coordinate 10.51,51.76
Scale 1:2578705
Magnifier 100%
Rotation 0.0 °
Render
EPSG:4326

WWU MÜNSTER

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, do in-between:

- read the data from the file as an array
- print this array