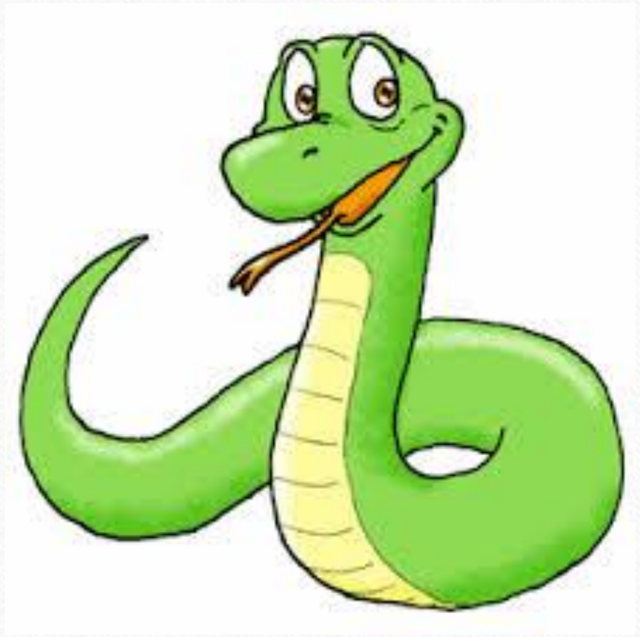
10. Spatial data processing



https://pcjericks.github.io/py-gdalogr-cookbook/

https://www.python.org/

Open Source RS/GIS modules

- Spatial data are modeled as raster and vector data type
- Spatial data can be accessed in Python through the use of different libraries
- Libraries are collections of reusable bits of code
- Libraries in Python are usually called modules. They can contain a variety of related things, including functions, constants, and classes.
- GDAL (Geospatial Data Abstraction Library) and the OGR Simple Features Library are Open Source RS/GIS libraries that can access raster and vector data

Open Source RS/GIS modules

- Development started by Frank Warmerdam in 1998. Now officially maintained by OSGEO.
- OGR Simple Features Library
  - Vector data access
  - Part of GDAL
- GDAL – Geospatial Data AbstractionLibrary
  - Raster data access
  - Used by commercial software like ArcGIS
  - Actually, C++ library, but Python bindings exist

Related modules

Numeric

- Sophisticated array manipulation (extremely useful for raster data!)
- This is the one we'll be using in class

NumPy

- Next generation of Numeric

Documentation

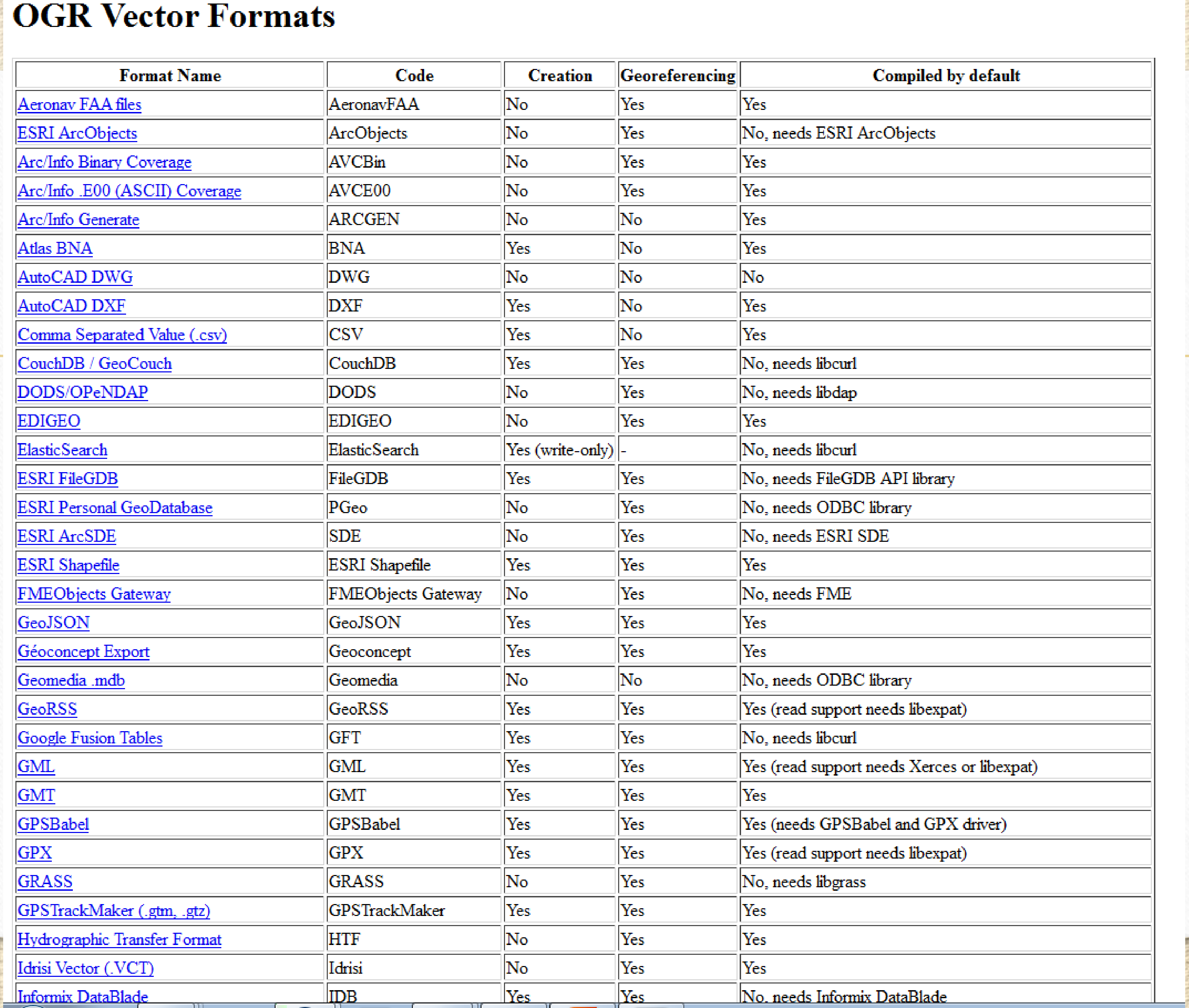
- GDAL: http://www.gdal.org/, gdal.py,

gdalconst.py (in the fwtools/pymod folder)

- OGR: http://www.gdal.org/ogr/, ogr.py
- Numeric: http://numpy.scipy.org/#older_array
- NumPy: http://numpy.scipy.org/

OGR

- Supports many different vector formats
- ESRI formats such as shapefiles, personal geodatabases and ArcSDE
- Other software such as MapInfo, GRASS, Microstation
- Open formats such as TIGER/Line, SDTS,GML, KML
- Databases such as MySQL, PostgreSQL,Oracle Spatial, Informix, ODBC



OGR Vector Formats

Format Name	Code	Creation	Georeferencing	Compiled by default
Aeronav FAA files	AeronavFAA	No	Yes	Yes
ESRI ArcObjects	ArcObjects	No	Yes	No, needs ESRI ArcObjects
Arc/Info Binary Coverage	AVCBin	No	Yes	Yes
Arc/Info .E00 (ASCII) Coverage	AVCE00	No	Yes	Yes
Arc/Info Generate	ARCGEN	No	No	Yes
Atlas BNA	BNA	Yes	No	Yes
AutoCAD DWG	DWG	No	No	No
AutoCAD DXF	DXF	Yes	No	Yes
Comma Separated Value (.csv)	CSV	Yes	No	Yes
CouchDB / GeoCouch	CouchDB	Yes	Yes	No, needs libcurl
DODS/OPeNDAP	DODS	No	Yes	No, needs libdap
EDIGEO	EDIGEO	No	Yes	Yes
ElasticSearch	ElasticSearch	Yes (write-only)	-	No, needs libcurl
ESRI FileGDB	FileGDB	Yes	Yes	No, needs FileGDB API library
ESRI Personal GeoDatabase	PGeo	No	Yes	No, needs ODBC library
ESRI ArcSDE	SDE	No	Yes	No, needs ESRI SDE
ESRI Shapefile	ESRI Shapefile	Yes	Yes	Yes
FMEObjects Gateway	FMEObjects Gateway	No	Yes	No, needs FME
GeoJSON	GeoJSON	Yes	Yes	Yes
Géoconcept Export	Geoconcept	Yes	Yes	Yes
Geomedia .mdb	Geomedia	No	No	No, needs ODBC library
GeoRSS	GeoRSS	Yes	Yes	Yes (read support needs libexpat)
Google Fusion Tables	GFT	Yes	Yes	No, needs libcurl
GML	GML	Yes	Yes	Yes (read support needs Xerces or libexpat)
GMT	GMT	Yes	Yes	Yes
GPSBabel	GPSBabel	Yes	Yes	Yes (needs GPSBabel and GPX driver)
GPX	GPX	Yes	Yes	Yes (read support needs libexpat)
GRASS	GRASS	No	Yes	No, needs libgrass
GPSTrackMaker (.gtm, .gtz)	GPSTrackMaker	Yes	Yes	Yes
Hydrographic Transfer Format	HTF	No	Yes	Yes
Idrisi Vector (.VCT)	Idrisi	No	Yes	Yes
Informix DataBlade	IDB	Yes	Yes	No, needs Informix DataBlade

Class Overview

- Geometry : The geometry classes (OGRGeometry, etc) encapsulate the OpenGIS model vector data as well as providing some geometry operations, and translation to/from well known binary(wkb) and text(wkt) format. A geometry includes a spatial reference system (projection).
- Spatial Reference : An OGRSpatialReference encapsulates the definition of a projection and datum.
- Feature : The OGRFeature encapsulates the definition of a whole feature, that is a geometry and a set of attributes.
- Feature Class Definition : The OGRFeatureDefn class captures the schema (set of field definitions) for a group of related features (normally a whole layer).

Class Overview

- Layer : OGRLayer is an abstract base class represent a layer of features in an GDALDataset.
- Dataset : A GDALDataset is an abstract base class representing a file or database containing one or more OGRLayer objects.
- Drivers : A GDALDriver represents a translator for a specific format, opening GDALDataset objects. All available drivers are managed by the GDALDriverManager.

Reading and Writing Vector Data with OGR: Importing OGR

- With FWTools:

```
import ogr
```

- With an OSGeo distribution:

```
from osgeo import ogr
```

- Handle both cases like this:

```
try:
    from osgeo import ogr
except:
    import ogr
```

OGR data drivers

- A driver is an object that knows how to interact with a certain data type (such as a shapefile)
- Need an appropriate driver in order to read or write data (need it explicitly for write)

OGR data drivers

- Might as well grab the driver for read operations so it is available for writing

1. Import the OGR module
2. Use **ogr.GetDriverByName(<driver_code>)**

```
import ogr
driver = ogr.GetDriverByName('ESRI Shapefile')
```

```
import ogr
cnt = ogr.GetDriverCount()
for i in range(cnt):
    driver = ogr.GetDriver(i)
    print (driver.GetName())
```

Opening a DataSource

- The Driver **Open()** method returns a DataSource object
- **Open(<filename>, <update>)**

  where <update> is 0 for read-only, 1 for writeable

```
fn = 'f:/data/classes/python/data/sites.shp'
dataSource = driver.Open(fn, 0)
if  dataSource is None:
    print ('Could not open ' + fn)
    sys.exit(1) #exit with an error code
```

Detour: Working directory

- Usually need to specify entire path for filenames
- Instead, set working directory with **os.chdir(<directory_path>)**

```
import ogr, sys, os
os.chdir('f:/data/classes/python/data')
driver = ogr.GetDriverByName('ESRI Shapefile')
dataSource = driver.Open('sites.shp', 0)
```

Opening a layer (shapefile)

- Use **GetLayer(<index>)** on a DataSource to get a Layer object
  - <index> is always 0 and optional for shapefiles
  - <index> is useful for other data types such as GML, TIGER

```
layer = dataSource.GetLayer()
layer = dataSource.GetLayer(0)
```

Getting info about the layer

- Get the number of features in the layer

```
numFeatures = layer.GetFeatureCount()
print ('Feature count: ' + str(numFeatures))
print ('Feature count:', numFeatures)
```

- Get the extent as a tuple (sort of a nonmodifiable list)

```
extent = layer.GetExtent()
print ('Extent:', extent)
print ('UL:', extent[0], extent[3])
print ('LR:', extent[1], extent[2])
```

Getting features

- If knows the FID (offset) of a feature, can use **GetFeature(<index>)** on the Layer

  feature = layer.GetFeature(0)

- Or can loop through all of the features

```
feature = layer.GetNextFeature()
while feature:
    # do something here
    feature = layer.GetNextFeature()
layer.ResetReading() #need if looping again from first feature
```

Getting a feature's attributes

- Feature objects have a **GetField(<name>)** method which returns the value of that attribute field
- There are variations, such as

  GetFieldAsString(<name>) and

  GetFieldAsInteger(<name>)

  id = feature.GetField('id')

  id = feature.GetFieldAsString('id')

Getting a feature’s geometry

- Feature objects have a method called **GetGeometryRef()** which returns a Geometry object (could be Point, Polygon, etc)
- Point objects have **GetX() GetY()** methods

```
geometry = feature.GetGeometryRef()
x = geometry.GetX()
y = geometry.GetY()
```

Destroying objects

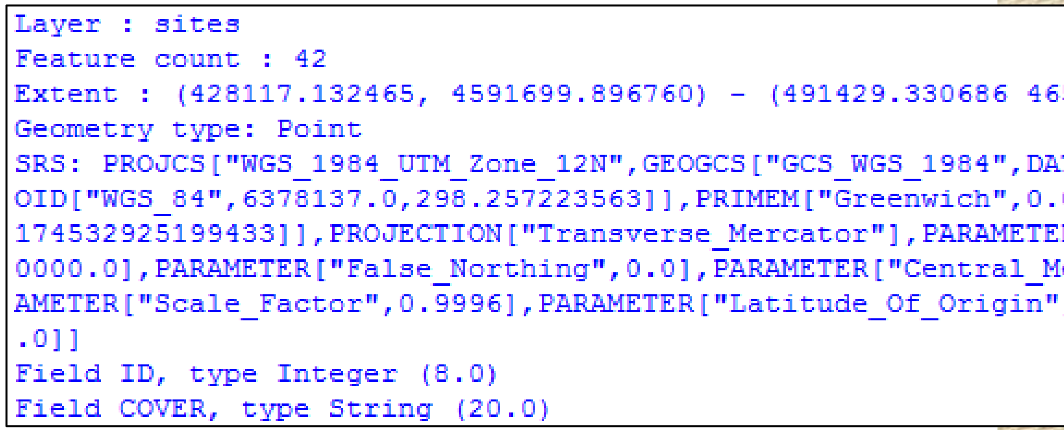
- For memory management purposes we need to make sure that we get rid of things such as features when done with them
- **feature.Destroy()**
- Also need to close DataSource objects when done with them
- **dataSource.Destroy()**

Counting features

```
# script to count features

# import modules
import ogr, os, sys

# set the working directory
os.chdir('f:/data/classes/python/data')

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

# open the data source
datasource = driver.Open('sites.shp', 0)
if datasource is None:
   print 'Could not open file'
   sys.exit(1)

# get the data layer
layer = datasource.GetLayer()

# loop through the features and count them
cnt = 0
feature = layer.GetNextFeature()
while feature:
   cnt = cnt + 1
   feature.Destroy()
   feature = layer.GetNextFeature()
print 'There are ' + str(cnt) + ' features'

# close the data source
datasource.Destroy()
```

Getting information :number of features, spatial extent in (xmin, ymin) – (xmax, ymax) format, coordinate system, list of attributes with their name, type and width.precision

```python
from osgeo import ogr
ds = ogr.Open('E:\\WRC\\2015winter\\Gis With Python\\labs\\pyFiles\\gggddaall\\data\\sites.shp')
for lyr in ds:
    print("Layer : %s" % lyr.GetName())
    print("Feature count : %d" % lyr.GetFeatureCount())
    (xmin, xmax, ymin, ymax) = lyr.GetExtent() # not in same order as ogrinfo
    print("Extent : (%f, %f) - (%f %f)" % (xmin, ymin, xmax, ymax))
    print("Geometry type: %s" % ogr.GeometryTypeToName(lyr.GetGeomType()))
    srs = lyr.GetSpatialRef()
    if srs is not None:
        print("SRS: %s"% srs.ExportToWkt())
        lyr_defn = lyr.GetLayerDefn()
        for i in range(lyr_defn.GetFieldCount()):
            field_defn = lyr_defn.GetFieldDefn(i)
            name = field_defn.GetName()
            type = ogr.GetFieldTypeName(field_defn.GetType())
            width = field_defn.GetWidth()
            prec = field_defn.GetPrecision()
            print('Field %s, type %s (%d.%d)' % (name, type, width, prec))
```



```
Layer : sites
Feature count : 42
Extent : (428117.132465, 4591699.896760) - (491429.330686 46
Geometry type: Point
SRS: PROJCS["WGS_1984_UTM_Zone_12N",GEOGCS["GCS_WGS_1984",DA
OID["WGS_84",6378137.0,298.257223563]],PRIMEM["Greenwich",0.
174532925199433]],PROJECTION["Transverse_Mercator"],PARAMETE
0000.0],PARAMETER["False_Northing",0.0],PARAMETER["Central_M
AMETER["Scale_Factor",0.9996],PARAMETER["Latitude_Of_Origin"
.0]]
Field ID, type Integer (8.0)
Field COVER, type String (20.0)
```

Exercise (assignment)

- Read coordinates and attributes from a shapefile
- Loop through the points in sites.shp
- Write out id, x & y coordinates, and cover type for each point to a text file, one point per line
- Hint: The two attribute fields in the shapefile are called "id" and "cover“

  : You can print more than one item on a line by separating them with commas, like this: print id, x, y, cover
- Mail source code and the output text file

Writing data

1. (a) Get or (b) create a writeable layer
2. Add fields if necessary
3. Create a feature
4. Populate the feature
5. Add the feature to the layer
6. Close the layer

1. (a) Getting a writeable layer

- Open an existing DataSource for writing and get the layer out of it

```
fn = 'f:/data/classes/python/data/sites.shp'
dataSource = driver.Open(fn, 1)
if dataSource is None:
    print 'Could not open ' + fn
    sys.exit(1) #exit with an error code
layer = dataSource.GetLayer(0)
```

1. (b) Creating a writeable layer

Create a new DataSource and Layer

I. CreateDataSource(<filename>) on a Driver object – the file cannot already exist!

II. CreateLayer(<name>,geom_type=<OGRwkbGeometryType>, [srs]) on a DataSource object

```
ds = driver.CreateDataSource('test.shp')

layer = ds.CreateLayer('test', geom_type=ogr.wkbPoint)
```

1. (b). (I) Checking if a datasource exists

- Use the exists(<filename>) method in the os.path module
- Use DeleteDataSource(<filename>) on a Driver object to delete it (this causes an error if the file does not exist)

```
import os
if os.path.exists('test.shp'):
    driver.DeleteDataSource('test.shp')
```

2. Adding fields (a)Getting FieldDefn

- Shapefiles need at least one attribute field
- Need a FieldDefn object first
- Copy one from an existing feature with

GetFieldDefnRef(<field_index>) or

GetFieldDefnRef(<field_name>)

```
fieldDefn = feature.GetFieldDefnRef(0)   #or
fieldDefn = feature.GetFieldDefnRef('id')
```

2. (b) Creating FieldDefn and Creat fields

- create a new FieldDefn with **FieldDefn(<field_name>, <OGRFieldType>)**, where the field name has a 12-character limit

```
fldDef = ogr.FieldDefn('id', ogr.OFTInteger)
```

- If it is a string field, set the width

```
fieldDefn = ogr.FieldDefn('id', ogr.OFTString)
fieldDefn.SetWidth(4)
```

- Now **create a field** on the layer using the FieldDefn object and **CreateField(<FieldDefn>)**

```
layer.CreateField(fieldDefn)
```

3. Creating new features

- Need a FeatureDefn object first
- Get it from the layer ***after*** adding any fields

```
featureDefn = layer.GetLayerDefn()
```

- Now use the FeatureDefn object to create a new Feature object

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

3. Creating new features

- Set the geometry for the new feature

```
point = ogr.Geometry(ogr.wkbPoint)
point.SetPoint(0,10,10)
feature.SetGeometry(point)
```

- Set the attributes with **SetField(<name>,<value>)**

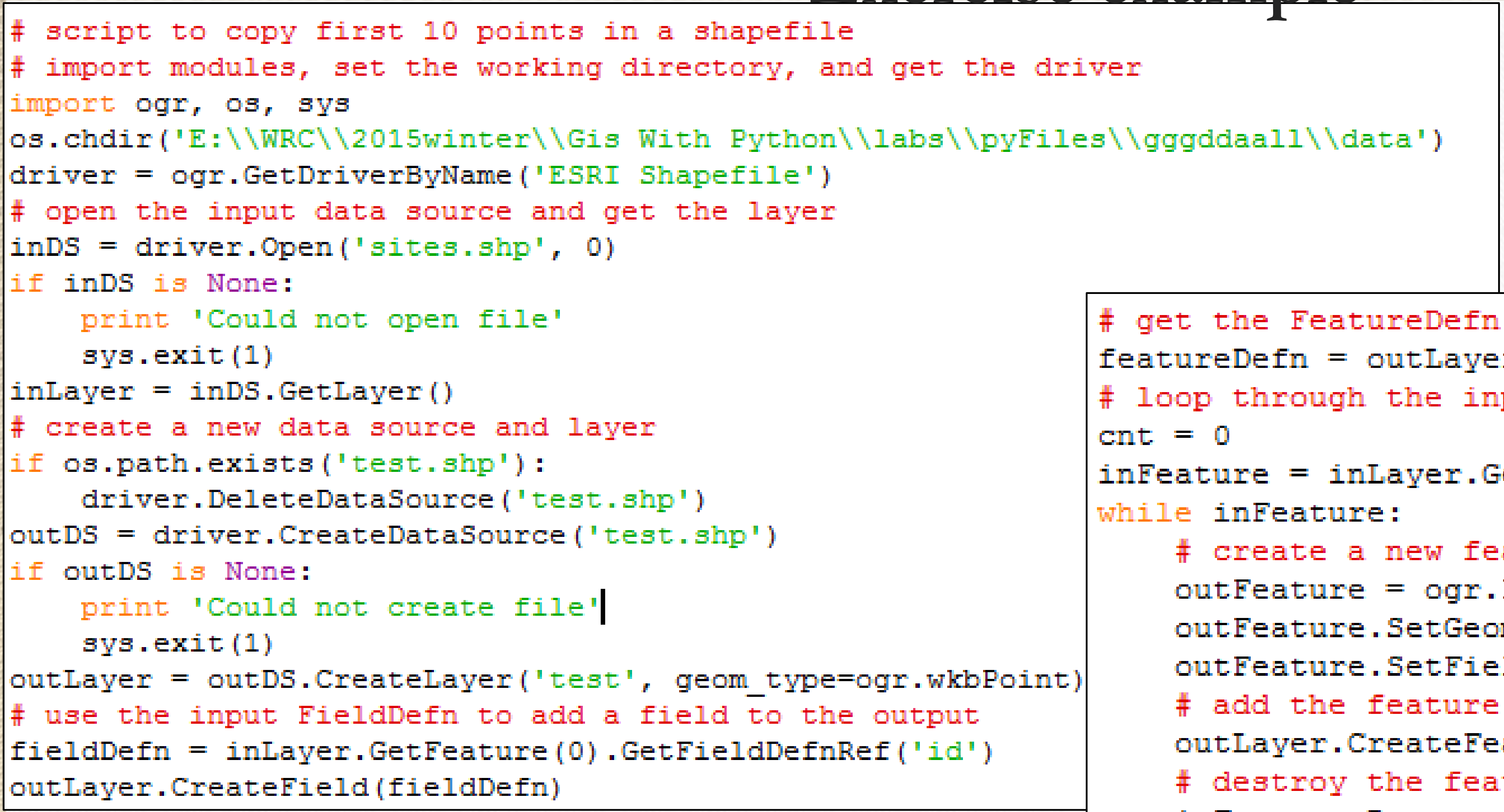
```
feature.SetField('id', 23)  # 4. populate the feature
```

- Write the feature to the layer

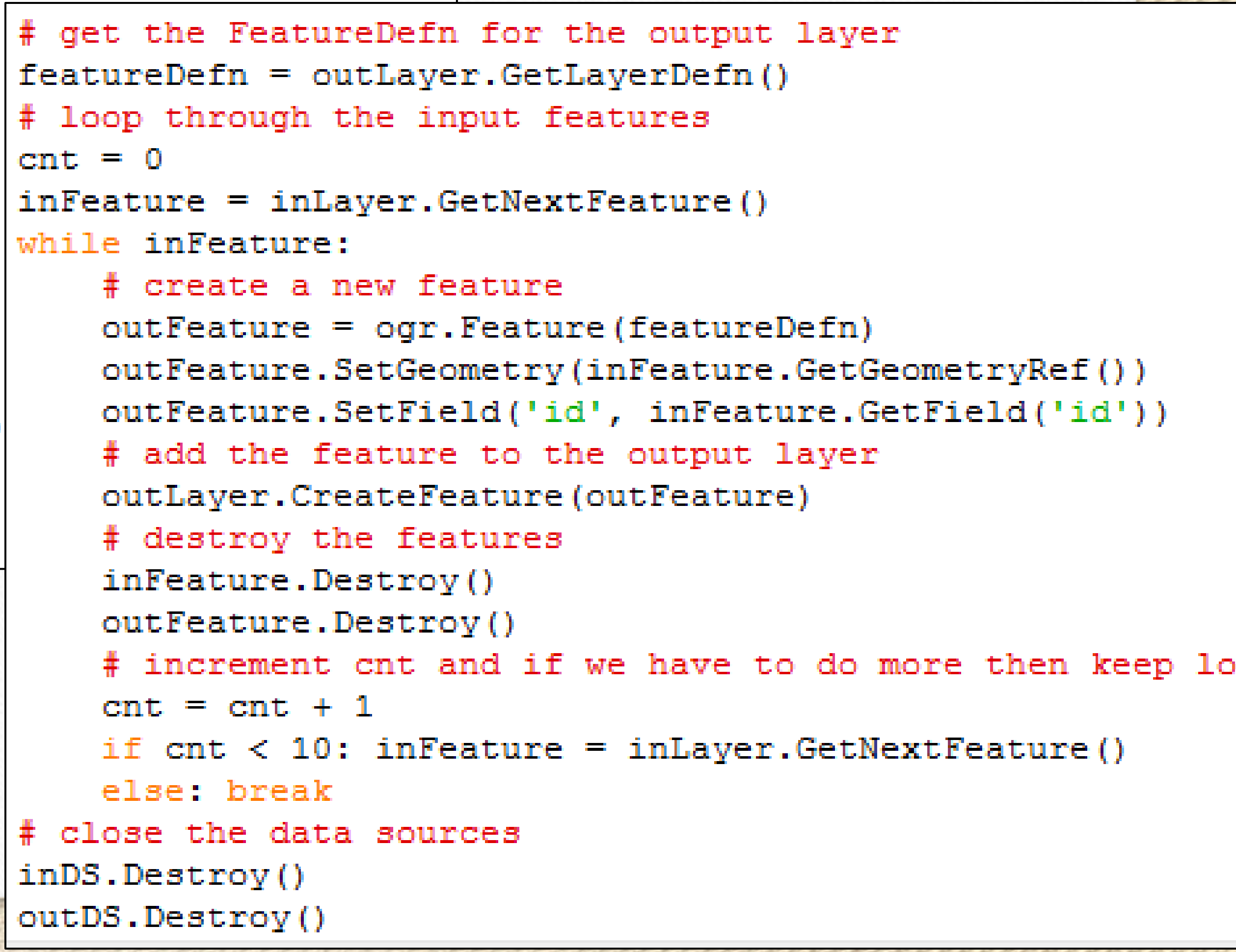
```
layer.CreateFeature(feature)  # 5. add feature to the layer
```

- Make sure to close the DataSource with **Destroy()** at the end so things get written

Exercise example



```python
# script to copy first 10 points in a shapefile
# import modules, set the working directory, and get the driver
import ogr, os, sys
os.chdir('E:\\WRC\\2015winter\\Gis With Python\\labs\\pyFiles\\gggddaall\\data')
driver = ogr.GetDriverByName('ESRI Shapefile')
# open the input data source and get the layer
inDS = driver.Open('sites.shp', 0)
if inDS is None:
    print 'Could not open file'
    sys.exit(1)
inLayer = inDS.GetLayer()
# create a new data source and layer
if os.path.exists('test.shp'):
    driver.DeleteDataSource('test.shp')
outDS = driver.CreateDataSource('test.shp')
if outDS is None:
    print 'Could not create file'
    sys.exit(1)
outLayer = outDS.CreateLayer('test', geom_type=ogr.wkbPoint)
# use the input FieldDefn to add a field to the output
fieldDefn = inLayer.GetFeature(0).GetFieldDefnRef('id')
outLayer.CreateField(fieldDefn)
```



```python
# get the FeatureDefn for the output layer
featureDefn = outLayer.GetLayerDefn()
# loop through the input features
cnt = 0
inFeature = inLayer.GetNextFeature()
while inFeature:
    # create a new feature
    outFeature = ogr.Feature(featureDefn)
    outFeature.SetGeometry(inFeature.GetGeometryRef())
    outFeature.SetField('id', inFeature.GetField('id'))
    # add the feature to the output layer
    outLayer.CreateFeature(outFeature)
    # destroy the features
    inFeature.Destroy()
    outFeature.Destroy()
    # increment cnt and if we have to do more then keep lo
    cnt = cnt + 1
    if cnt < 10: inFeature = inLayer.GetNextFeature()
    else: break
# close the data sources
inDS.Destroy()
outDS.Destroy()
```

Exercise example

```
# script to copy first 10 points in a shapefile
# import modules, set the working directory, and get the driver
import ogr, os, sys
os.chdir('E:\\WRC\\2015winter\\Gis With Python\\labs\\data')
driver = ogr.GetDriverByName('ESRI Shapefile')
# open the input data source and get the layer
inDS = driver.Open('sites.shp', 0)
if inDS is None:
    print 'Could not open file'
    sys.exit(1)
inLayer = inDS.GetLayer()
# create a new data source and layer
if os.path.exists('test.shp'):
    driver.DeleteDataSource('test.shp')
outDS = driver.CreateDataSource('test.shp')
if outDS is None:
    print 'Could not create file'
    sys.exit(1)
outLayer = outDS.CreateLayer('test', geom_type=ogr.wkbPoint)
# use the input FieldDefn to add a field to the output
fieldDefn = inLayer.GetFeature(0).GetFieldDefnRef('id')
outLayer.CreateField(fieldDefn)
# get the FeatureDefn for the output layer
featureDefn = outLayer.GetLayerDefn()
# loop through the input features
cnt = 0
inFeature = inLayer.GetNextFeature()
while inFeature:
    # create a new feature
    outFeature = ogr.Feature(featureDefn)
    outFeature.SetGeometry(inFeature.GetGeometryRef())
    outFeature.SetField('id', inFeature.GetField('id'))
    # add the feature to the output layer
    outLayer.CreateFeature(outFeature)
    # destroy the features
    inFeature.Destroy()
    outFeature.Destroy()
    # increment cnt and if we have to do more then keep looping
    cnt = cnt + 1
    if cnt < 10: inFeature = inLayer.GetNextFeature()
    else: break
# close the data sources
inDS.Destroy()
outDS.Destroy()
```

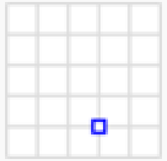
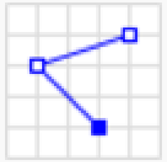
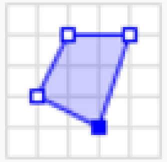
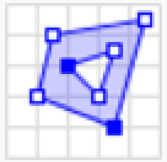
Exercise(assignment)

- Copy selected features from one shapefile to another
    - Create a new point shapefile and add an ID field
    - Loop through the points in sites.shp
    - If the cover attribute for a point is ‘trees’ then write that point out to the new shapefile
- Mail the source code and a screenshot of the new shapefile being displayed

Geometry

- **Geometry** : The geometry classes (OGRGeometry, etc) encapsulate the OpenGIS model vector data as well as providing some geometry operations, and translation to/from well known binary(wkb) and text(wkt) format. A geometry includes a spatial reference system (projection).

Types of geometry include OGRPoint, OGRLineString, OGRPolygon, OGRGeometryCollection, OGRMultiPolygon, OGRMultiPoint, and OGRMultiLineString.

Type	Examples	
Point		`POINT (30 10)`
LineString		`LINESTRING (30 10, 10 30, 40 40)`
Polygon		`POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))`
		`POLYGON ((35 10, 45 45, 15 40, 10 20, 35 10), (20 30, 35 35, 30 20, 20 30))`

Geometry

OGC Simple Features

- Point
- LineString
- Polygon
- MultiPoint
- MultiLineString
- MultiPolygon
- GeometryCollection

Creating a new Geometry

1. Create an empty Geometry object with **ogr.Geometry(<OGRwkbGeometryType>)**
2. Define what the geometry is in a different way for each type (point, line, polygon, etc.)

Creating points

- Use `AddPoint( <x>, <y>, [<z>])` to set coordinates. The height value <z> is optional and defaults to 0

```
point = ogr.Geometry(ogr.wkbPoint)

point.AddPoint(10,20)
```

```
from osgeo import ogr
point = ogr.Geometry(ogr.wkbPoint)
point.AddPoint(1198054.34, 648493.09)
print point
```

```
POINT (1198054.34 648493.09 0)
```

Creating lines

- Add new vertices to the line with **AddPoint(<x>, <y>, [<z>])**
- Change the coordinates of a vertex with **SetPoint(<index>, <x>, <y>, [<z>])**
- where <index> is the index of the vertex to change

```
line = ogr.Geometry(ogr.wkbLineString)
line.AddPoint(10,10)
line.AddPoint(20,20)
line.SetPoint(0,30,30) #(10,10) -> (30,30)
print line
```

Creating lines

- To get the number of vertices in a line use `GetPointCount()`

```
print line.GetPointCount()
```

- To get the x,y coordinates for a specific vertex use `GetX(<vertex_index>)` and `GetY(<vertex_index>)`

```
print line.GetX(0)

print line.GetY(0)
```

Creating polygons

- Must create rings first and then add them to the polygon later
- To make a ring, create an empty ring Geometry and then use AddPoint(<x>,<y>) to add vertices

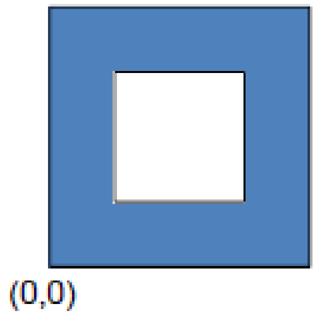
```
ring = ogr.Geometry(ogr.wkbLinearRing)
ring.AddPoint(0,0)
ring.AddPoint(100,0)
ring.AddPoint(100,100)
ring.AddPoint(0,100)
```

- To close the ring, use **CloseRings()** or make the last vertex the same as the first

```
ring.CloseRings()

ring.AddPoint(0,0)
```

Creating polygons

```
outring = ogr.Geometry(ogr.wkbLinearRing)
outring.AddPoint(0,0)
outring.AddPoint(100,0)
outring.AddPoint(100,100)
outring.AddPoint(0,100)
outring.AddPoint(0,0)
inring = ogr.Geometry(ogr.wkbLinearRing)
inring.AddPoint(25,25)
inring.AddPoint(75,25)
inring.AddPoint(75,75)
inring.AddPoint(25,75)
inring.CloseRings()
```



Creating polygons

- Now make the polygon and add the rings with **`AddGeometry(<geometry>)`**

```
polygon = ogr.Geometry(ogr.wkbPolygon)
polygon.AddGeometry(outring)
polygon.AddGeometry(inring)
```

- Get the number of rings in a polygon with **`GetGeometryCount()`**

```
print polygon.GetGeometryCount()
```

Creating polygons

- Get a ring object from a polygon with **`GetGeometryRef(<ring_index>)`**; indices are the same order you added them to thepolygon

```
    outring = polygon.GetGeometryRef(0)
    inring = polygon.GetGeometryRef(1)
```

- • Get the number of vertices in a ring and the coordinates of those vertices as inring.GetPointCount(), inring.GetX(<vertex_index>) and inring.GetY(<vertex_index>)

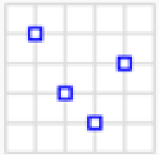
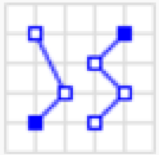
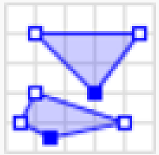
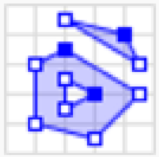
```
from osgeo import ogr
# Create outer ring
outRing = ogr.Geometry(ogr.wkbLinearRing)
outRing.AddPoint(1154115.274565847, 686419.4442701361)
outRing.AddPoint(1154115.274565847, 653118.2574374934)
outRing.AddPoint(1165678.1866605144, 653118.2574374934)
outRing.AddPoint(1165678.1866605144, 686419.4442701361)
outRing.AddPoint(1154115.274565847, 686419.4442701361)

# Create inner ring
innerRing = ogr.Geometry(ogr.wkbLinearRing)
innerRing.AddPoint(1149490.1097279799, 691044.6091080031)
innerRing.AddPoint(1149490.1097279799, 648030.5761158396)
innerRing.AddPoint(1191579.1097525698, 648030.5761158396)
innerRing.AddPoint(1191579.1097525698, 691044.6091080031)
innerRing.AddPoint(1149490.1097279799, 691044.6091080031)
```

```
# Create polygon
poly = ogr.Geometry(ogr.wkbPolygon)
poly.AddGeometry(outRing)
poly.AddGeometry(innerRing)

#count vertices in ring and Get X of  first vertex
outring=poly.GetGeometryRef(0)
print outring.GetPointCount()
print outring.GetX(0)

print poly.ExportToWkt()
```

```
5

1154115.27457

POLYGON ((1154115.2745658469 686419.44427013607 0,11541
0,1165678.1866605144 653118.25743749342 0,1165678.18666051
686419.44427013607 0),(1149490.1097279799 691044.6091080030
0,1191579.1097525698 648030.57611583965 0,1191579.10975256
691044.60910800309 0))
```

Multi Geometries

- MultiPoint, MultiLineString, MultiPolygon…
- Create a geometry and then add it to the multi-version with **AddGeometry(<geom>)**

Type	Examples	
MultiPoint		`MULTIPOINT ((10 40), (40 30), (20 20), (30 10))`
		`MULTIPOINT (10 40, 40 30, 20 20, 30 10)`
MultiLineString		`MULTILINESTRING ((10 10, 20 20, 10 40), (40 40, 30 30, 40 20, 30 10))`
MultiPolygon		`MULTIPOLYGON (((30 20, 45 40, 10 40, 30 20)), ((15 5, 40 10, 10 20, 5 10, 15 5)))`
		`MULTIPOLYGON (((40 40, 20 45, 45 30, 40 40)), ((20 35, 10 30, 10 10, 30 5, 45 20, 20 35), (30 20, 20 15, 20 25, 30 20)))`

Create a MultiPoint

```
from osgeo import ogr
multipoint = ogr.Geometry(ogr.wkbMultiPoint)
point1 = ogr.Geometry(ogr.wkbPoint)
point1.AddPoint(1251243.7361610543,
598078.7958668759)
multipoint.AddGeometry(point1)
point2 = ogr.Geometry(ogr.wkbPoint)
point2.AddPoint(1240605.8570339603,
601778.9277371694)
multipoint.AddGeometry(point2)
point3 = ogr.Geometry(ogr.wkbPoint)
point3.AddPoint(1250318.7031934808,
606404.0925750365)
multipoint.AddGeometry(point3)
print multipoint.ExportToWkt()
```

Create a MultiLineString

```
from osgeo import ogr
multiline =
ogr.Geometry(ogr.wkbMultiLineString)
line1 = ogr.Geometry(ogr.wkbLineString)
line1.AddPoint(1214242.4174581182,
617041.9717021306)
line1.AddPoint(1234593.142744733,
629529.9167643716)
multiline.AddGeometry(line1)
line1 = ogr.Geometry(ogr.wkbLineString)
line1.AddPoint(1184641.3624957693,
626754.8178616514)
line1.AddPoint(1219792.6152635587,
606866.6090588232)
multiline.AddGeometry(line1)
print multiline.ExportToWkt()
```

Create a MultiPolygon

```
from osgeo import ogr
multipolygon = ogr.Geometry(ogr.wkbMultiPolygon)
# Create ring #1
ring1 = ogr.Geometry(ogr.wkbLinearRing)
ring1.AddPoint(1204067.0548148106,
634617.5980860253)
ring1.AddPoint(1204067.0548148106,
620742.1035724243)
ring1.AddPoint(1215167.4504256917,
620742.1035724243)
ring1.AddPoint(1215167.4504256917,
634617.5980860253)
ring1.AddPoint(1204067.0548148106,
634617.5980860253)
# Create polygon #1
poly1 = ogr.Geometry(ogr.wkbPolygon)
poly1.AddGeometry(ring1)
multipolygon.AddGeometry(poly1)
```

```
# Create ring #2
ring2 = ogr.Geometry(ogr.wkbLinearRing)
ring2.AddPoint(1179553.6811741155, 647105.5431482664)
ring2.AddPoint(1179553.6811741155, 626292.3013778647)
ring2.AddPoint(1194354.20865529, 626292.3013778647)
ring2.AddPoint(1194354.20865529, 647105.5431482664)
ring2.AddPoint(1179553.6811741155, 647105.5431482664)
# Create polygon #2
poly2 = ogr.Geometry(ogr.wkbPolygon)
poly2.AddGeometry(ring2)
multipolygon.AddGeometry(poly2)
print multipolygon.ExportToWkt()
```

Create a GeometryCollection

```
from osgeo import ogr
# Create a geometry collection
geomcol = ogr.Geometry(ogr.wkbGeometryCollection)
# Add a point
point = ogr.Geometry(ogr.wkbPoint)
point.AddPoint(-122.23, 47.09)
geomcol.AddGeometry(point)
# Add a line
line = ogr.Geometry(ogr.wkbLineString)
line.AddPoint(-122.60, 47.14)
line.AddPoint(-122.48, 47.23)
geomcol.AddGeometry(line)
print geomcol.ExportToWkt()
```

Do not destroy geometries that come from an existing feature. Destroy geometries that are created during script execution

Exercise(example)

```
# script to add a point to a new shapefile
# import modules and set the working directory
import ogr, os, sys
os.chdir('E:\\WRC\\data')
# get the driver
driver = ogr.GetDriverByName('ESRI Shapefile')
# create a new data source and layer
if os.path.exists('test.shp'):
    driver.DeleteDataSource('test.shp')
ds = driver.CreateDataSource('test.shp')
if ds is None:
    print 'Could not create file'
    sys.exit(1)
layer = ds.CreateLayer('test',
geom_type=ogr.wkbPoint)
```

```
# add an id field to the output
fieldDefn = ogr.FieldDefn('id', ogr.OFTInteger)
layer.CreateField(fieldDefn)
# create a new point object
point = ogr.Geometry(ogr.wkbPoint)
point.AddPoint(150, 75)
# get the FeatureDefn for the output layer
featureDefn = layer.GetLayerDefn()
# create a new feature
feature = ogr.Feature(featureDefn)
feature.SetGeometry(point)
feature.SetField('id', 1)
# add the feature to the output layer
layer.CreateFeature(feature)
# destroy the geometry and feature and close the
point.Destroy()
feature.Destroy()
ds.Destroy()
```

Projections

- The **OGRSpatialReference** class is intended to store an OpenGIS Spatial Reference System definition.
- Currently local, geographic and projected coordinate systems are supported
- The spatial coordinate system data model is inherited from the OpenGIS Well Known Text format

Projections

Lots of different ways to specify projections

- **Well Known Text (WKT):** http://en.wikipedia.org/wiki/Well-known_text
- **PROJ.4:** www.remotesensing.org/geotiff/proj_list/
- **EPSG** (European Petroleum Survey Group): see epsg file in your FWTools2.x.x/proj_lib directory
- **USGS**: see importFromUSGS() description at www.gdal.org/ogr/classOGRSpatialReference.html
- **ESRI .prj** (import only), PCI software, XML

Projections

ESRI .prj

```
PROJCS["WGS_1984_UTM_Zone_12N",GEOGCS["GCS_WGS_1984",DATUM["D_
WGS_1984",SPHEROID["WGS_1984",6378137,298.257223563]],PRIMEM["
Greenwich",0],UNIT["Degree",0.0174532925199433295]],PROJECTION[
"Transverse_Mercator"],PARAMETER["False_Easting",500000],PARAM
ETER["False_Northing",0],PARAMETER["Central_Meridian",-
111],PARAMETER["Scale_Factor",0.9996],PARAMETER["Latitude_Of_O
rigin",0],UNIT["Meter",1]]
```

WKT

```
PROJCS["UTM Zone 12, Northern
Hemisphere",GEOGCS["WGS_1984",DATUM["WGS_1984",SPHEROID["WGS
84",6378137,298.2572235630016],TOWGS84[0,0,0,0,0,0,0]],PRIMEM[
"Greenwich",0],UNIT["degree",0.0174532925199433],AUTHORITY["EP
SG","4326"]],PROJECTION["Transverse_Mercator"],PARAMETER["lati
tude_of_origin",0],PARAMETER["central_meridian",-
111],PARAMETER["scale_factor",0.9996],PARAMETER["false_easting
",500000],PARAMETER["false_northing",0],UNIT["Meter",1],AUTHOR
ITY["EPSG","32612"]]
```

Projections

Pretty WKT

```
PROJCS["UTM Zone 12, Northern Hemisphere",
    GEOGCS["WGS_1984",
        DATUM["WGS_1984",
            SPHEROID["WGS 84",6378137,298.2572235630016],
            TOWGS84[0,0,0,0,0,0,0]],
        PRIMEM["Greenwich",0],
        UNIT["degree",0.0174532925199433],
        AUTHORITY["EPSG","4326"]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",0],
    PARAMETER["central_meridian",-111],
    PARAMETER["scale_factor",0.9996],
    PARAMETER["false_easting",500000],
    PARAMETER["false_northing",0],
    UNIT["Meter",1],
    AUTHORITY["EPSG","32612"]]
```

Projections

EPSG

```
32612
```

Proj.4

```
+proj=utm +zone=12 +ellps=WGS84 +datum=WGS84 +units=m
+no_defs
```

USGS

```
(1, 12, (0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0), 12)
```

PCI

```
('UTM 12 D000', 'METRE', (0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0))
```

Getting a layer's projection

- If a data set has projection information stored with it (for example, a .prj file with a shapefile)

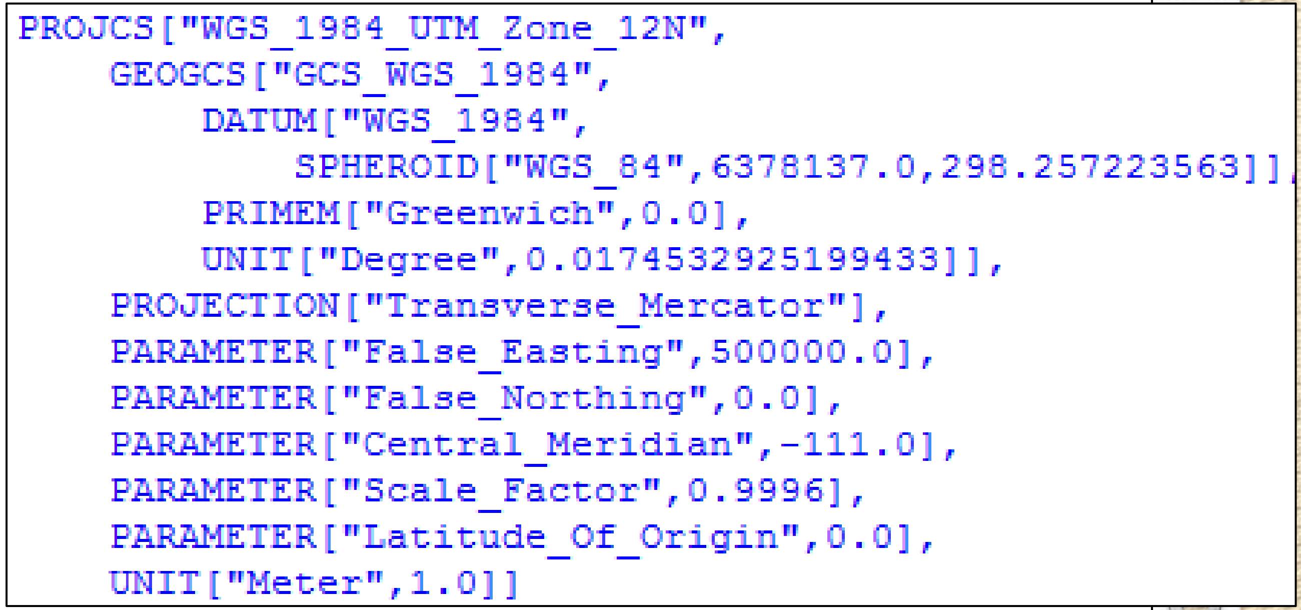
  spatialRef = layer.GetSpatialRef()

- If no projection information with the data, then **GetSpatialRef()** returns **None**
- Can also get it from a geometry object:

  spatialRef = geom.GetSpatialReference()

Getting a layer's projection

```
from osgeo import ogr, osr
driver = ogr.GetDriverByName('ESRI Shapefile')
dataset = driver.Open(r'E:\WRC\data\sites.shp')
# from Layer
layer = dataset.GetLayer()
spatialRef  = layer.GetSpatialRef()
print spatialRef
# from Geometry
feature = layer.GetNextFeature()
geom = feature.GetGeometryRef()
spatialRef  = geom.GetSpatialReference()
print spatialRef
```



```
PROJCS["WGS_1984_UTM_Zone_12N",
    GEOGCS["GCS_WGS_1984",
        DATUM["WGS_1984",
            SPHEROID["WGS_84",6378137.0,298.257223563]]
        PRIMEM["Greenwich",0.0],
        UNIT["Degree",0.0174532925199433]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["False_Easting",500000.0],
    PARAMETER["False_Northing",0.0],
    PARAMETER["Central_Meridian",-111.0],
    PARAMETER["Scale_Factor",0.9996],
    PARAMETER["Latitude_Of_Origin",0.0],
    UNIT["Meter",1.0]]
```

Creating a new projection

1. Import osr
2. Create an empty SpatialReference object with **osr.SpatialReference()**
3. Use one of the import methods (next slide) to import projection information into the SpatialReference object

```
from osgeo import osr
spatialRef = osr.SpatialReference()
spatialRef.ImportFromEPSG(32612) # from EPSG
```

or
spatialRef.ImportFromProj4('+proj=utm +zone=12 +ellps=WGS84 +datum=WGS84 +units=m +no_defs')

- **`ImportFromWkt(<wkt>)`**
- **`ImportFromEPSG(<epsg>)`**
- **`ImportFromProj4(<proj4>)`**
- **`ImportFromESRI(<proj_lines>)`**
- **`ImportFromPCI(<proj>, <units>,<parms>)`**
- **`ImportFromUSGS(<proj_code>, <zone>)`**
- **`ImportFromXML(<xml>)`**

Exporting a projection

These methods will return strings

- ExportToWkt()
- ExportToPrettyWkt()
- ExportToProj4()
- ExportToPCI()
- ExportToUSGS()
- ExportToXML()

```
from osgeo import ogr, osr
driver = ogr.GetDriverByName('ESRI Shapefile')
dataset = driver.Open(r'E:\WRC\data\sites.shp')
# from Layer
layer = dataset.GetLayer()
spatialRef  = layer.GetSpatialRef()
#check any one of the export
print spatialRef.ExportToWkt()
print spatialRef.ExportToPrettyWkt()
print spatialRef.ExportToPCI()
print spatialRef.ExportToUSGS()
print spatialRef.ExportToXML()
print spatialRef
```

Creating an ESRI .prj file

```
from osgeo import ogr, osr
spatialRef = osr.SpatialReference()
spatialRef.ImportFromEPSG(26912)
spatialRef.MorphToESRI()
file = open('yourshpfile.prj', 'w')
file.write(spatialRef.ExportToWkt())
file.close()
```

Projecting a geometry

1. Create a CoordinateTransformation
    1. Get the SpatialReference for the source
    2. Get the SpatialReference for the target
    3. Create the CoordinateTransformation with
        `osr.CoordinateTransformation(<sourceSpatialRef>,<targetSpatialRef>)`
2. Use **`Transform(<CoordTransform> )`** on the geometry object

In order to project an entire DataSource, need to project one geometry at a time

Projecting a geometry

```
from osgeo import ogr
from osgeo import osr
source = osr.SpatialReference()
source.ImportFromEPSG(2927)
target = osr.SpatialReference()
target.ImportFromEPSG(4326)
transform = osr.CoordinateTransformation(source, target)
point = ogr.CreateGeometryFromWkt("POINT (1120351.57 741921.42)")
point.Transform(transform)
print point.ExportToWkt()
```

Projecting a geometry

```
import ogr, osr, os
os.chdir(r'E:\WRC\data')
driver = ogr.GetDriverByName('ESRI Shapefile')
dataset = driver.Open('Settlements.shp')
layer = dataset.GetLayer()
feature = layer.GetNextFeature()
geom = feature.GetGeometryRef()
print geom.GetX(), geom.GetY()

geoSR = osr.SpatialReference()
geoSR.ImportFromEPSG(4326) # unprojected WGS84

utmSR =osr.SpatialReference()
utmSR.ImportFromEPSG(32645) # UTM 45N WGS84

coordTrans =osr.CoordinateTransformation(geoSR, utmSR)
geom.Transform(coordTrans)

print geom.GetX(), geom. GetY()
```

```
87.6139831543 27.5187206268
560635.126486 3044040.56118
```

Exercise (try)

Reproject a shapefile

- Create a new point shapefile
- Loop through the points in Settlements.shp, reproject each one, and write it out to the new shapefile
- Go from EPSG 4326 (unprojected WGS84) to EPSG 32645 (UTM 45N WGS84)