Filters & Simple Analysis of Vector Data; Functions & Modules

Open Source RS/GIS Python
Week 3
Attribute filters

• The Layer object has a method `SetAttributeFilter(<where_clause>)`

• Resets reading so `GetNextFeature()` gets the first feature that matches the filter

• Pass `None` to clear the filter

```python
>>> layer.GetFeatureCount()
42
>>> layer.SetAttributeFilter("cover = 'shrubs'")
>>> layer.GetFeatureCount()
6
>>> layer.SetAttributeFilter(None)
>>> layer.GetFeatureCount()
42
```
Spatial filters

- There are two methods on the Layer object to set spatial filters
  - `SetSpatialFilter(<geom>)`
  - `SetSpatialFilterRect(<minx>, <miny>, <maxx>, <maxy>)`
- Use `SetSpatialFilter(None)` to clear the filter
layerAreas is polygon and layerSites is points

```python
>>> featAreas = layerAreas.GetNextFeature()
>>> poly = featAreas.GetGeometryRef()

>>> layerSites.GetFeatureCount()
42
>>> layerSites.SetSpatialFilter(poly)

>>> layerSites.GetFeatureCount()
3

>>> layerSites.SetSpatialFilterRect(460000, 4590000, 490000, 4600000)

>>> layerSites.GetFeatureCount()
4

>>> layerSites.SetSpatialFilter(None)

>>> layerSites.GetFeatureCount()
42
```
More complicated filters

• DataSource objects have a method `ExecuteSQL(<SQL>)`

• Returns a Layer object that can be looped through like other layers

• When done with the resulting layer, call `ReleaseResultSet(<result_layer>)` on the DataSource

• See http://www.gdal.org/ogr/ogr_sql.html for more info about valid SQL syntax
To select all features where cover type is 'grass' and to return them in descending order:

```python
result = dsSites.ExecuteSQL("select * from sites where cover = 'grass' order by id desc")
resultFeat = result.GetNextFeature()
while resultFeat:
    print resultFeat.GetField('id')
    resultFeat = result.GetNextFeature()
```

```
42
40
:
4
```
• To count the number of features where cover type is 'grass':

```python
>>> result = dsSites.ExecuteSQL("select count(*)
   from sites where cover = 'grass'")

>>> result.GetFeatureCount()
1

>>> result.GetFeature(0).GetField(0)
11

>>> dsSites.ReleaseResultSet(result)
```
To get a list of unique cover types:

```python
result = ds.ExecuteSQL("select distinct cover from sites")
resultFeat = result.GetNextFeature()
while resultFeat:
    print resultFeat.GetField(0)
    resultFeat = result.GetNextFeature()
```

shrubs


trees

rocks

grass

bare

Water
To get a count of each cover type:

```python
coverLayer = ds.ExecuteSQL('select distinct cover from sites')
coverFeat = coverLayer.GetNextFeature()
while coverFeat:
    cntLayer = ds.ExecuteSQL("select count(*) from sites where cover = '{0}' + coverFeat.GetField(0) + '{0}"")
    print coverFeat.GetField(0) + ' ' + cntLayer.GetFeature(0).GetFieldAsString(0)
    ds.ReleaseResultSet(cntLayer)
    coverFeat = coverLayer.GetNextFeature()

ds.ReleaseResultSet(coverLayer)

shrubs 6
trees 11
rocks 6
grass 11
bare 6
water 2
```
Intersect

• Do two geometries intersect?

```python
>>> poly2.Intersect(poly1)
0
>>> poly2.Intersect(poly3)
1
>>> poly2.Intersect(poly2)
1
>>> poly1.Intersect(ptA)
0
>>> poly1.Intersect(ptB)
1
>>> poly1.Intersect(line)
1
>>> poly3.Intersect(line)
1
>>> line.Intersect(ptB)
1
```
Disjoint

• Are two geometries disjoint?

```python
>>> poly2.Disjoint(poly1)
1
>>> poly2.Disjoint(poly3)
0
>>> poly1.Disjoint(ptA)
1
>>> poly1.Disjoint(ptB)
0
>>> poly1.Disjoint(line)
0
>>> poly3.Disjoint(line)
0
>>> line.Disjoint(ptB)
```
Touches

• Do two geometries touch (on the edge)?

```python
>>> poly2.Touches(poly1)
0
>>> poly2.Touches(poly3)
0
>>> poly1.Touches(line)
0
>>> poly1.Touches(ptB)
0
>>> poly3.Touches(line)
1
```
Crosses

• Do two geometries cross each other?

```python
>>> poly2.Crosses(poly1)
0
>>> poly2.Crosses(poly3)
0
>>> poly2.Crosses(line)
1
>>> poly3.Crosses(line)
0
>>> poly1.Crosses(line)
1
>>> line.Crosses(ptB)
0
```
Within

• Is one geometry within another one?

```python
>>> poly3.Within(poly2)
0
>>> line.Within(poly2)
0
>>> ptA.Within(poly1)
0
>>> ptB.Within(poly1)
1
>>> poly1.Within(ptB)
0
```
Contains

- Does one geometry contain another one?

```python
>>> poly3.Contains(poly2)
0
>>> line.Contains(poly2)
0
>>> poly2.Contains(line)
0
>>> poly1.Contains(ptA)
0
>>> poly1.Contains(ptB)
1
>>> ptB.Contains(poly1)
0
```
Overlaps

- Do two geometries overlap?

```python
>>> poly2.Overlaps(poly1)
0
>>> poly2.Overlaps(poly3)
1
>>> poly2.Overlaps(line)
0
>>> poly3.Overlaps(line)
0
>>> poly1.Overlaps(ptB)
0
```
Simple geoprocessing

poly3.Intersection(poly2)

poly3.Union(poly2)

poly3.Difference(poly2)

poly3.SymmetricDifference(poly2)
• `<geom>.Buffer(<distance>)`
  • Buffer a geometry, returns a new geometry
• `<geom1>.Equal(<geom2>)`
  • Are the geometries equal?
• `<geom1>.Distance(<geom2>)`
  • Returns the shortest distance between the two geometries
• `<geom>.GetEnvelope()`
  • Returns the geometry's extent as a list (minx, maxx, miny, maxy)
Assignment 3a

• Use filters and buffers
  • Use an attribute filter to restrict cache_towns.shp to Nibley ("name" field)
  • Buffer the Nibley geometry by 1500
  • Use the new geometry with a spatial filter on sites.shp to find all sites within 1500 meters of Nibley
  • Print out the "id" value for those sites
  • Turn in your code and a screenshot of what got printed
Defining functions

- Writing functions allows easy reuse of code
- The keyword `def` means you’re about to define a function
- Provide the function name and parameters
- Use `return` to return a value

```python
def add(n1, n2):  #function 'add' takes 2 args
    return n1 + n2
```
For example, a function to reproject a shapefile (using EPSG codes) might look something like this:

```python
def ProjectShapefile(inFN, outFN, inEPSG, outEPSG):
    # open inFN and create empty outFN
    # create CoordinateTransform using EPSG codes
    # loop through features in inFN
        # get feature geometry
        # reproject geometry
        # write geometry to outFN
        # destroy features
    # close files
```
Keywords & default parameters

• User can pass parameters by passing them in order (the usual way) or by using the variable name as a keyword

```python
add(n2=5, n1=3)
```

• Can provide a default value by setting it in the function declaration line

```python
def printNumbers(start=0, stop=10, step=1):
```
def printNumbers(start=0, stop=10, step=1):
    for i in range(start, stop, step):
        print i,

>>> printNumbers()
0 1 2 3 4 5 6 7 8 9
>>> printNumbers(5)
5 6 7 8 9
>>> printNumbers(5,15)
5 6 7 8 9 10 11 12 13 14
>>> printNumbers(5,15,2)
5 7 9 11 13
>>> printNumbers(step=2, stop=15, start=5)
5 7 9 11 13
>>> printNumbers(step=2)
0 2 4 6 8
>>> printNumbers(5, step=2)
5 7 9
Exceptions

• “Catch” exceptions to implement custom error handling, like flushing a buffer to disk before dying
• Can have as many `except` statements as you need per `try` statement

```python
try:
    # code here
except <exception_name>:
    # code to handle error here

try:
    # code here
finally:
    # code that always runs
```
• Have to put the `try...except` statement inside of a `try...finally` statement if you want both behaviors

```python
try:
    try:
        # code here
    except <exception_name>:
        # code to handle error here
    except <exception_name>:
        # code to handle error here
finally:
    # code that always runs
```
import sys, traceback

def getValue(array, index):
    try:  # wrap everything so some code always executes
        try:  # wrap code that might cause an error
            return array[index]
        except IndexError:
            # catch exc if index out of range
            print 'Looks like a bad index'
        except:
            # catch something we didn't anticipate
            print 'Exception type:', sys.exc_info()[0]
            print 'Exception value:', sys.exc_info()[1]
            traceback.print_exc()
    finally:
        # good place to close files, db connections...
        print 'This code always runs!'
>>> test = ['a', 'b', 'c']

>>> print getValue(test, 0)  # good index
This code always runs!
a
>>> print getValue(test, 4)  # bad index
Looks like a bad index
This code always runs!
None

>>> print getValue(test, 'a')  # unknown error
Exception type: exceptions.TypeError
Exception value: list indices must be integers
Traceback (most recent call last):
  File "exceptions.py", line 6, in getValue
    return array[index]
TypeError: list indices must be integers
This code always runs!
None
Creating modules

• Modules are handy places to keep functions
• Just put the function in a file called <name>.py
• A file named mymod.py could be imported using
  import mymod
• And a function inside it called myFunction could be called like
  mymod.myFunction()
• Import other needed modules at the top of your module
import math

def add(n1, n2):
    return n1 + n2

def printNumbers(start=0, stop=10, step=1):
    for i in range(start, stop, step):
        print i,

def maxNumFromBits(bits):
    return int(math.pow(2, bits) - 1)
• Assuming that the module in the previous slide is saved as mymod.py, it can be used like this:

```python
>>> import mymod
>>> mymod.add(3, 2)
5
>>> mymod.printNumbers(5, 8)
5 6 7
>>> mymod.maxNumFromBits(8)
255
```
Finding modules

• Python will look for a module in the directory that the running script is in
• Then the PYTHONPATH environment variable
• Possibly the current working directory (depends on platform & version)
• Then standard library directories (i.e. site-packages)
• Modify `sys.path` to change the search path (but only for the current python session)

• If you delete things from `sys.path` you may not be able to import other modules, so you should probably just append

```python
>>> import testmod # can't find it
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ImportError: No module named testmod
>>> import sys
>>> sys.path.append('d:/temp')
>>> import testmod # now it worked
>>> testmod.add(2,3)
5
```
Glob

• Lists files in a directory that match a pattern
  • * matches multiple characters
  • ? matches 1 character
  • [] matches character ranges, like [0-9], [a-z], or [a,e,i,o,u]

```python
>>> import glob
>>> glob.glob('./[0-9].*')
['./1.gif', './2.txt']
>>> glob.glob('*.gif')
['1.gif', 'card.gif']
>>> glob.glob('?.gif')
['1.gif']
```
• For example, to loop through all .shp files in a directory:

```python
>>> import glob
>>> for fn in glob.glob('d:/data/*.*.shp'):
...   print fn
...
```

```output
d:/data/classes/python/data\areas.shp
d:/data/classes/python/data\lines.shp
d:/data/classes/python/data\points.shp
d:/data/classes/python/data\sites.shp
d:/data/classes/python/data\ut_counties.shp
```
Manipulating filenames

```python
import os
fn = 'c:/data/test.shp'

• To get the basename of a filename:
  >>> os.path.basename(fn)
  'test.shp'

• To get the basename with no extension:
  >>> os.path.splitext(os.path.basename(fn))[0]
  'test'
  >>> os.path.basename(fn)[:-4]
  'test'
```
• An easy way to add a suffix to a filename:

```python
>>> newfn = fn.replace('.shp', '_proj.shp')
>>> newfn
'c:/data/test_proj.shp'
```

• An easy way to change the extension on a filename:

```python
>>> newfn = fn.replace('.shp', '.prj')
>>> newfn
'c:/data/test.prj'
```
Homework

• Write a function to reproject a shapefile using EPSG codes
  • User should pass in the input & output filenames and input & output EPSG codes
  • Put the function in a module
• Write a script that imports the new module and uses the function to reproject all of the shapefiles in this week’s data
  • Go from EPSG 26912 (UTM 12N NAD83) to EPSG 4269 (unprojected NAD83)
# function to copy fields (not the data) from one
# layer to another
# parameters:
#   fromLayer: layer object that contains the fields
to copy
#   toLayer: layer object to copy the fields into

def copyFields(fromLayer, toLayer):
    featureDefn = fromLayer.GetLayerDefn()
    for i in range(featureDefn.GetFieldCount()):
        toLayer.CreateField(featureDefn.GetFieldDefn(i))
# function to copy attributes from one feature to another
# this assumes the features have the same attribute fields!
# parameters:
# fromFeature: feature object that contains the data to copy
# toFeature: feature object that the data is to be copied into

def copyAttributes(fromFeature, toFeature):
    for i in range(fromFeature.GetFieldCount()):
        fieldName = fromFeature.GetFieldDefnRef(i).GetName()
        toFeature.SetField(fieldName, fromFeature.GetField(fieldName))