Python
Working with Feature Data
Ghislain Prince
Jibin Liu
Python: Working with Feature Data

- Join us as we discuss working with feature data in ArcGIS using ArcPy and the data access module (arcpy.da). Highlights and demonstrations will include getting the best performance out of cursors, editing data, working with NumPy arrays and 3rd party libraries to extend analysis, and managing geodata.
Python: Working with Raster Data
Mesquite B

The integration of map algebra with Python opens a new dimension for raster analysis and the automation of geoprocessing workflows. Using a real world example, this session will include an introduction to the Spatial Analyst ArcPy module, best practices for using the raster object and classes to expand your modeling capability, using raster functions to optimize performance, and using NumPy arrays to extend your analysis.

Categories - - Esri Technical Session, Geoprocessing
Slides and demos

https://github.com/jibin-geoprocessing/WorkingWithFeatureData
<table>
<thead>
<tr>
<th>OBJECTID</th>
<th>Shape</th>
<th>CITY_NAME</th>
<th>CNTRY_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Point</td>
<td>Cuiaba</td>
<td>Brazil</td>
</tr>
<tr>
<td>2</td>
<td>Point</td>
<td>Brasilia</td>
<td>Brazil</td>
</tr>
<tr>
<td>3</td>
<td>Point</td>
<td>Goiania</td>
<td>Brazil</td>
</tr>
<tr>
<td>4</td>
<td>Point</td>
<td>Campo Grande</td>
<td>Brazil</td>
</tr>
<tr>
<td>5</td>
<td>Point</td>
<td>Pedro Juan Caballero</td>
<td>Paraguay</td>
</tr>
<tr>
<td>6</td>
<td>Point</td>
<td>Salto del Guaira</td>
<td>Paraguay</td>
</tr>
</tbody>
</table>
Cursors

- Cursors provide record-by-record, feature-by-feature access
  - Basic necessity for many workflows

<table>
<thead>
<tr>
<th>Cursor</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>SearchCursor</td>
<td>Read-only access</td>
</tr>
<tr>
<td>UpdateCursor</td>
<td>Update or delete rows</td>
</tr>
<tr>
<td>InsertCursor</td>
<td>Insert rows</td>
</tr>
</tbody>
</table>
- Detailed documentation
- Required arguments: table, fields
- Optional arguments: many, all there for optimization of your code

```python
fields = ['field1', 'field2']
cursor = arcpy.da.InsertCursor(table, fields)
cursor.insertRow([1, 10])
```
“Classic” cursors

- Two implementations
  - arcpy.da cursors and “Classic” cursors (accessible directly on the arcpy namespace)
  - Which one? Unless you have legacy code you don’t want to update, use arcpy.da

- “Classic” cursors
  - For scripts written before 10.1
  - Use Row objects
  - Row values are handled using setValue, getValue properties

```python
# Example code

cursor = arcpy.InsertCursor(table)
row = cursor.newRow()
row.setValue("field1", 1)
row.setValue("field2", 10)
cursor.insertRow(row)
```
with statements

- arcpy.da Cursors support with statements

```python
with arcpy.da.SearchCursor(table, field) as cursor:
    for row in cursor:
        print row[0]
```

- with statements
  - In Python, with statements provide context management
  - Locks *
  - Code clarity
```python
p = {u'Le xf3n Cortév9s': u'Le xf3n Cortév9s Castro',
     u'Vxelezque de Coronado': u'Vxelezque de Coronado'}

arcpy.da.UpdateCursor("canton",
   field_names = ("NAME_2", "Canton", "area_km2","pop_2008"),
   where_clause = u"NAME_2 IN ('le xf3n Cortxe9s', 'Vxelezque de Coronado')") as fc_cursor

for fc_row in fc_cursor:
    with arcpy.da.SearchCursor("canton_pop",
        field_names = ("Canton", "area_km2","pop_2008"),
        where_clause = u"Canton = "'{}'".format(lookup[fc_row[0]]) as tab_cursor:
        for tab_row in tab_cursor:
            print("fc_row : {}".format(fc_row))
            print("tab_row : {}".format(tab_row))
            fc_cursor.updateRow(fc_row[1] = list(tab_row))
```

Cursors

esriurl.com/10618
More on cursors

- Row values are accessed by index
- Good for performance, not as good for code readability

- Alternatively, can convert to a dictionary on the fly
  - Wrap with a generator function
  - Access by name
Fields and tokens

- For best performance, use only those fields you need
  - Can use "*" for all fields if necessary

- Tokens can be also be used
  - Get only what you need (accessing full geometry is more expensive)

```
'OID@'
'SHAPE@XY'
'SHAPE@TRUECENTROID'
'SHAPE@X'
'SHAPE@Y'
'SHAPE@Z'
'SHAPE@M'
'SHAPE@JSON'
'SHAPE@WKB'
'SHAPE@WKT'
'SHAPE@'
'SHAPE@AREA'
'SHAPE@LENGTH'
```
Working with geometry

- Creating geometry objects can be a bit unwieldly
- Many different options for accessing/creating geometry within a cursor

### Geometry objects

```python
• cursor = arcpy.da.InsertCursor(fc, 'SHAPE@')
• line = arcpy.Polyline(arcpy.Array([arcpy.Point(-7216000.0, 5944500.0),
                                        arcpy.Point(-7225700.0, 5934500.0)]),
                        arcpy.SpatialReference(3857))
• cursor.insertRow([line])
```

### Esri JSON

```python
• cursor = arcpy.da.InsertCursor(fc, 'SHAPE@JSON')
• json_line = {"paths": [[-7216000.0, 5944500.0], [-7225700.0, 5934500.0]],
              "spatialReference": {"wkid": 102100, "latestWkid": 3857}}
• cursor.insertRow([json.dumps(json_line)])
```

### List of coordinates

```python
• cursor = arcpy.da.InsertCursor(fc, 'SHAPE@')
• coordinate_list = [(-7216000.0, 5944500.0), (-7225700.0, 5934500.0)]
• cursor.insertRow([coordinate_list])
```
Editor class

- Uses edit sessions and edit operations to manage transactions
- Edits are temporary until saved and permanently applied
- Can quit an edit session without saving changes

When do you need to use?
- To edit feature classes that participate in a:
  - Topology
  - Geometric network
- Versioned datasets in enterprise geodatabases
- Some objects and feature classes with class extensions
Editor using a with statement

- Editor supports `with` statements
  - Handle appropriate start, stop and abort calls for you

```python
with arcpy.da.Editor(workspace) as edit:
    # your edits
```

Exception — operation is cancelled, edit session is closed without saving
No exceptions — stop the operation and save and close the edit session

Open an edit session and start an edit operation
Editor methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startEditing({with_undo}, {multiuser_mode})</td>
<td>Starts an edit session.</td>
</tr>
<tr>
<td>stopEditing(save_changes)</td>
<td>Stops an edit session.</td>
</tr>
<tr>
<td>startOperation()</td>
<td>Starts an edit operation.</td>
</tr>
<tr>
<td>stopOperation()</td>
<td>Stops an edit operation.</td>
</tr>
<tr>
<td>abortOperation()</td>
<td>aborts an edit operation.</td>
</tr>
<tr>
<td>undoOperation()</td>
<td>Undo an edit operation (roll back modifications).</td>
</tr>
<tr>
<td>redoOperation()</td>
<td>Redoes an edit operation.</td>
</tr>
</tbody>
</table>

# Start an edit session
```
edit = arcpy.da.Editor(workspace)
```

# Edit session is started without an undo/redo stack # for versioned data
```
edit.startEditing(False, True)
```

# Start an edit operation
```
edit.startOperation()
```

# Edits

# Stop the edit operation
```
edit.stopOperation()
```

# Stop the edit session and save changes
```
edit.stopEditing(True)
```
Iterating over data

- Many processes require cataloging or iterating over data
- Common theme are arcpy list functions
  - 30+ across arcpy and modules

<table>
<thead>
<tr>
<th>arcpy.da list functions:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ListDomains</td>
<td>Lists the attribute domains belonging to a geodatabase</td>
</tr>
<tr>
<td>ListFieldConflictFilters</td>
<td>Lists the fields in a versioned feature class that have field conflict filters applied</td>
</tr>
<tr>
<td>ListReplicas</td>
<td>Lists the replicas in a workspace</td>
</tr>
<tr>
<td>ListSubtypes</td>
<td>Return a dictionary of the subtypes for a table or feature class</td>
</tr>
<tr>
<td>ListVersions</td>
<td>List the versions in a workspace</td>
</tr>
</tbody>
</table>
Walk

- Traverse a directory structure to find ArcGIS data types
- Returns a tuple of three: path, path names, and filenames

```python
walk = arcpy.da.Walk(workspace, datatype=datatypes)

for path, path_names, data_names in walk:
    for data_name in data_names:
        do_something(os.path.join(path, data_name))
```

- Similar pattern to Python’s `os.walk`
- Comparison:
  - Walk: http://esriurl.com/5931
  - The hard way: http://esriurl.com/5932
Iterating over data

import pprint
r = arcpy.da.Walk(os.getcwd(), datatype="FeatureClass")
for dirpath, dirnames, filenames in r:
    for filename in filenames:
        fc = os.path.join(dirpath, filename)
        st = arcpy.da.ListSubtypes(fc)
        if len(st.keys()) > 1:
            pprint(st)
NumPy

- **NumPy** is a 3rd party Python library for scientific computing
  - A powerful array object
  - Sophisticated analysis capabilities

- **arcpy.da** supports converting tables and feature classes to/from numpy arrays
  - FeatureClassToNumPyArray, TableToNumPyArray

- Can also converting rasters to/from numpy arrays
  - RasterToNumPyArray, NumPyArrayToRaster
  - (found in main arcpy namespace)
NumPy functions

- `arcpy.da` provides additional support for tables and feature classes

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FeatureClassToNumPyArray</td>
<td>Convert a feature class to an array</td>
</tr>
<tr>
<td>TableToNumPyArray</td>
<td>Convert a table to an array</td>
</tr>
<tr>
<td>NumPyArrayToFeatureClass</td>
<td>Convert an array to a Feature Class</td>
</tr>
<tr>
<td>NumPyArrayToTable</td>
<td>Convert an array to a Table</td>
</tr>
<tr>
<td>ExtendTable</td>
<td>Join an array to a Table</td>
</tr>
</tbody>
</table>
Export to NumPy

- Can convert tables and feature classes into numpy arrays for further analysis

```python
import arcpy
import numpy

in_fc = "c:/data/usa.gdb/USA/counties"
field1 = "INCOME"
field2 = "EDUCATION"

array1 = arcpy.da.FeatureClassToNumPyArray(in_fc, [field1, field2])

# Print correlation coefficients for comparison of 2 fields
print numpy.corrcoef((array1[field1], array1[field2]))
```
Import from NumPy

- Take the product of your work in numpy and export it back to ArcGIS
  - Points only

```python
array1 = numpy.array([[1, (471316.3, 5000448.7)],
                      [2, (470402.4, 5000492.2)]],
                      numpy.dtype([('idfield', numpy.int32),
                                    ('XY', '<f8', 2)]))

sr = arcpy.SpatialReference(wkid)

# Export the numpy array to a feature class using the XY field to represent the output point feature
arcpy.da.NumPyArrayToFeatureClass(array1, outFC, ['XY'], sr)
```

- Polygons, lines, multipoints?
  - [http://esriurl.com/5862](http://esriurl.com/5862)
Pandas

- **Pandas** is 3rd party library known for high-performance, easy-to-use data structures and analysis tools
- Added to ArcMap, Server, Engine at 10.4
- Added to ArcGIS Pro 1.1

```python
def features_to_pandas(in_features, fields):
    in_arr = arcpy.da.FeatureClassToNumPyArray(in_features, fields)
    return pandas.DataFrame(in_arr)

def pandas_to_features(dataframe, output, geometry_columns, wkid=None):
    in_arr = dataframe.to_records()
    spatial_ref = '' if not wkid else arcpy.SpatialReference(wkid)
    return arcpy.da.NumPyArrayToFeatureClass(in_arr, output, geometry_columns,
                                             spatial_reference=spatial_ref)
```
y.da.InsertCursor(in_fc, ['SHAPE@', id_field]) as cursor
unique_array = numpy.unique(in_array[id_field])  # unique

rate through unique sets, get array that matches unique, convert coordinates to a list and insert via cursor
unique_value in unique_array:
    = in_array[in_array[id_field] == unique_value]

if len(a) >= min_xy_pairs:  # skip if not enough x,y pairs:
    cursor.insertRow([a[geom_fields].tolist()], unique_value)
else:
    pass  # skip if not enough x,y pairs
Please Take Our Survey!

Download the Esri Events app and find your event.

Select the session you attended.

Scroll down to the “Feedback” section.

Complete Answers, add a Comment, and Select “Submit.”
Slides and demos

https://github.com/jibin-geoprocessing/WorkingWithFeatureData