



Python Working with Feature Data

Ghislain Prince

Jibin Liu

2018 Esri DEVSummit Conference | Palm Springs, CA

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.

Not in this presentation

- **GP Tools**
- **Raster data**

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>

cities X

Field: Add Delete Calculate Selection: Zoom To

	OBJECTID	Shape	CITY_NAME	CNTRY_NAME
	1	Point	Cuiaba	Brazil
	2	Point	Brasilia	Brazil
	3	Point	Goiania	Brazil
	4	Point	Campo Grande	Brazil
	5	Point	Pedro Juan Caballero	Paraguay
	6	Point	Salto del Guaira	Paraguay

0 of *2000 selected Load All

Cursors

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

SearchCursor	Read-only access
UpdateCursor	Update or delete rows
InsertCursor	Insert rows

- Detailed documentation
 - <http://pro.arcgis.com/en/pro-app/arcpy/data-access/searchcursor-class.htm>
- Required arguments: table, fields
- Optional arguments: many, all there for optimization of your code

```
• 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

```
• 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

```
• 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

```

p = {u'Le\xfbn Cort\xe9s'      : u'Le\xfbn Cort\xe9s Castro',
      u'V\xe9isquez de Coronado' : u'V\xe9isquez de Coronado' }

arcpy.da.UpdateCursor("canton",
                      field_names = ("NAME_2", "Canton", "area_km2", "pop_2008"),
                      where_clause = u"NAME_2 IN ('Le\xfbn Cort\xe9s', 'V\xe9isquez 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

```
• with arcpy.da.SearchCursor(table, fields) as cursor:  
•     for row in cursor:  
•         print(row[17]) # index 17 is RoadName field
```

```
def row_as_dict(cursor):  
•     for row in cursor:  
•         yield dict(zip(cursor.fields, row))  
  
• with arcpy.da.SearchCursor(table, fields) as cursor:  
•     for row in row_as_dict(cursor):  
•         print(row['RoadName'])
```

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'
```

← Equivalent to using the geometry field name

Working with geometry

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

- **Geometry objects**

```
• 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**

```
• 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**

```
• 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

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

← Open an edit session and start an edit operation

↑ Exception—operation is cancelled, edit session is closed without saving

↑ No exceptions—stop the operation and save and close the edit session

Editor class

- Editor class also includes methods for working with edit sessions and operations

```
# 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)
```

Editor methods	
startEditing ({with_undo}, {multiuser_mode})	Starts an edit session.
stopEditing(save_changes)	Stops an edit session.
startOperation()	Starts an edit operation.
stopOperation()	Stops an edit operation.
abortOperation()	Aborts an edit operation.
undoOperation()	Undo an edit operation (roll back modifications).
redoOperation()	Redoes an edit operation.

Iterating over data

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

arcpy.da list functions:

ListDomains	Lists the attribute domains belonging to a geodatabase
ListFieldConflictFilters	Lists the fields in a versioned feature class that have field conflict filters applied
ListReplicas	Lists the replicas in a workspace
ListSubtypes	Return a dictionary of the subtypes for a table or feature class
ListVersions	List the versions in a workspace

Walk

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

```
• 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>

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

Iterating over data

esriurl.com/10618

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

Function	
<code>FeatureClassToNumPyArray</code>	Convert a feature class to an array
<code>TableToNumPyArray</code>	Convert a table to an array
<code>NumPyArrayToFeatureClass</code>	Convert an array to a Feature Class
<code>NumPyArrayToTable</code>	Convert an array to a Table
<code>ExtendTable</code>	Join an array to a Table

Export to NumPy

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

```
• 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

```
• array1 = numpy.array([(1, (471316.3, 5000448.7)),
•           (2, (470402.4, 5000049.2))],
•           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>


```
y.da.InsertCursor(in_fc, ['SHAPE@', id_field]) as cur
e_array = numpy.unique(in_array[id_field]) # unique

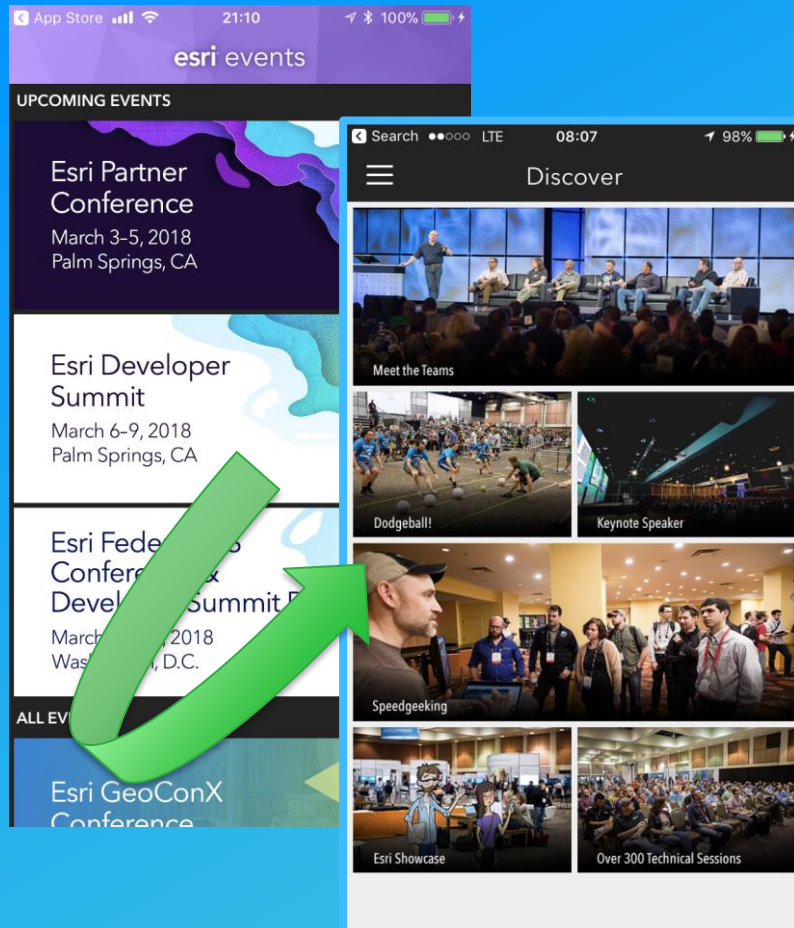
rate through unique sets, get array that matches unique
ue, convert coordinates to a list and insert via cursor
nique_value in unique_array:
    = in_array[in_array[id_field] == unique_value]
f len(a) >= min_xy_pairs: # skip if not enough x,y pairs
    cursor.insertRow([a[geom_fields].tolist(), unique_value])
lse:
    pass # skip if not enough x,y pairs
```

arcpy, numpy and pandas

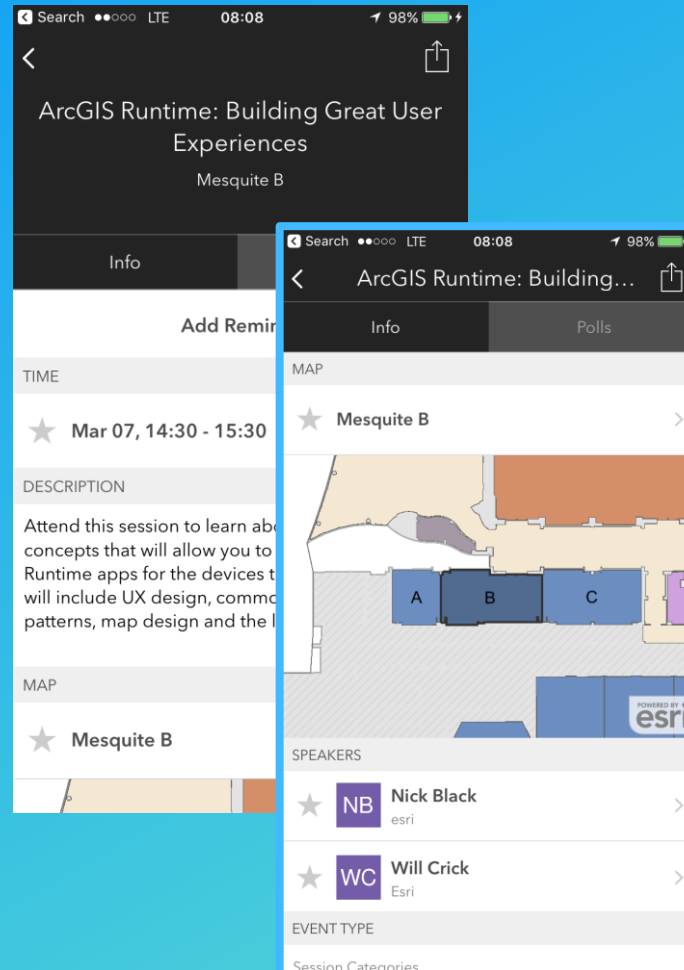
esriurl.com/10618

Please Take Our Survey!

Download the Esri Events app
and find your event

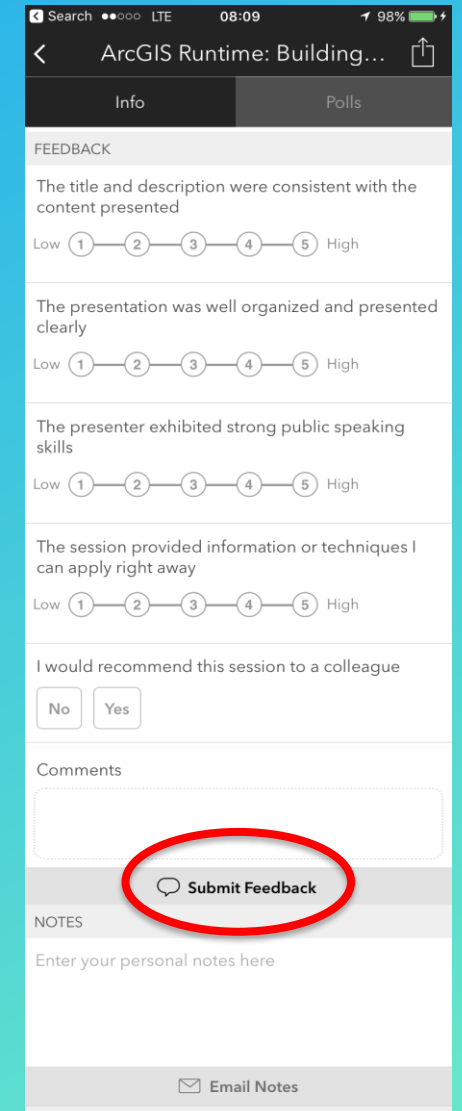


Select the session you
attended



Scroll down to the
“Feedback” section

Complete Answers,
add a Comment,
and Select “Submit”



Slides and demos

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



esri

THE
SCIENCE
OF
WHERE