Python
Working with Feature Data

Dave Wynne

Ghislain Prince

2019 ESRI DEVELOPER SUMMIT
Palm Springs, CA

Python: Working with Feature Data using ArcPy

Santa Rosa

Join us as we discuss working with feature data in ArcGIS using both ArcPy and the data access module (arcpy.da). Highlights and demonstrations will include getting the best performance, working with cursors, editing data, and managing geodata.

- **Materials at http://esriurl.com/10618**

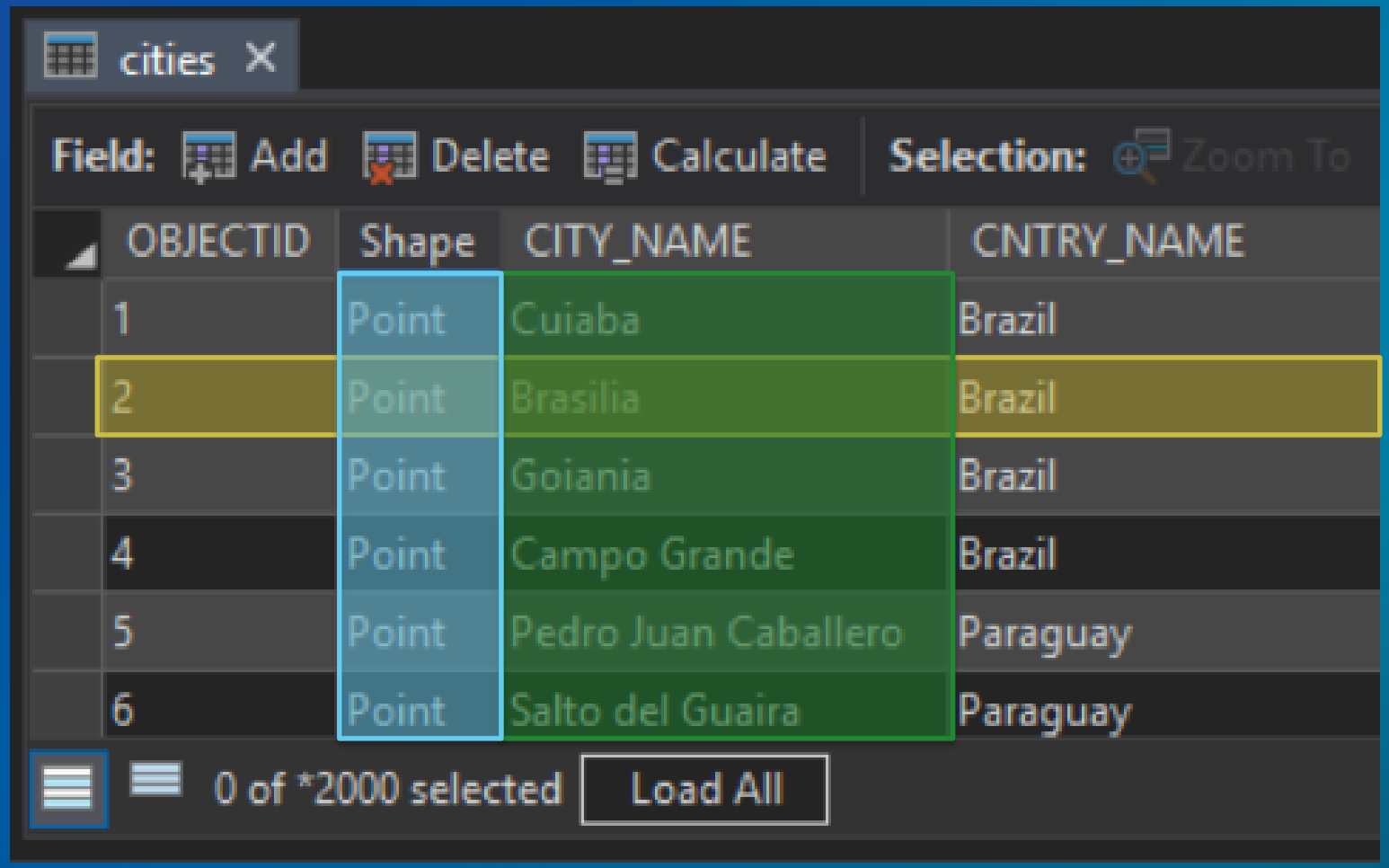
Cursors

- **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

- **Two implementations**
  - `arcpy.da` **cursors and "Classic" cursors**
  - **Which one? Unless you have legacy code you don't want to update, use** `arcpy.da`

Table basics



Cursors

- `arcpy.da` cursors use lists and tuples
  - Row values are accessed by index

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

- "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

Demo: Cursors

More on cursors

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

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

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

```
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
  - Although not recommended, you can use "*" for all fields if necessary
- Tokens can be also be used
  - Get only what you need (accessing full geometry is more expensive)

Equivalent to using the geometry field name

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

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**

```python
# 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.

Demo: Edit sessions

Working with geometry

- Creating geometry objects can be a bit unwieldly
- 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])
```

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`

Demo: Iterating over data

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	
FeatureClassToNumPyArray	Convert a feature class to an array
TableToNumPyArray	Convert a table to an array
NumPyArrayToFeatureClass	Convert an array to a Feature Class
NumPyArrayToTable	Convert an array to a Table
ExtendTable	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

pandas

- `pandas` is 3rd party library known for:
  - high-performance
  - easy-to-use data structures and
  - analysis tools

```
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)
```

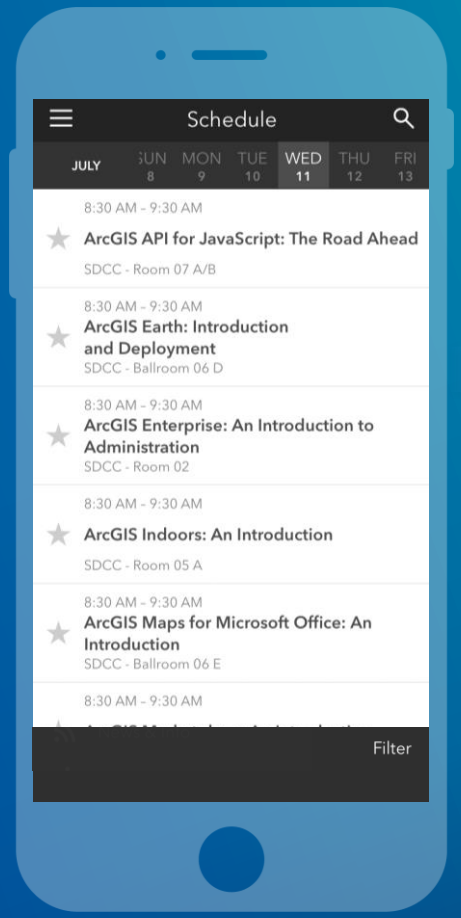
Demo: arcpy, numpy and pandas

esri | THE SCIENCE OF WHERE

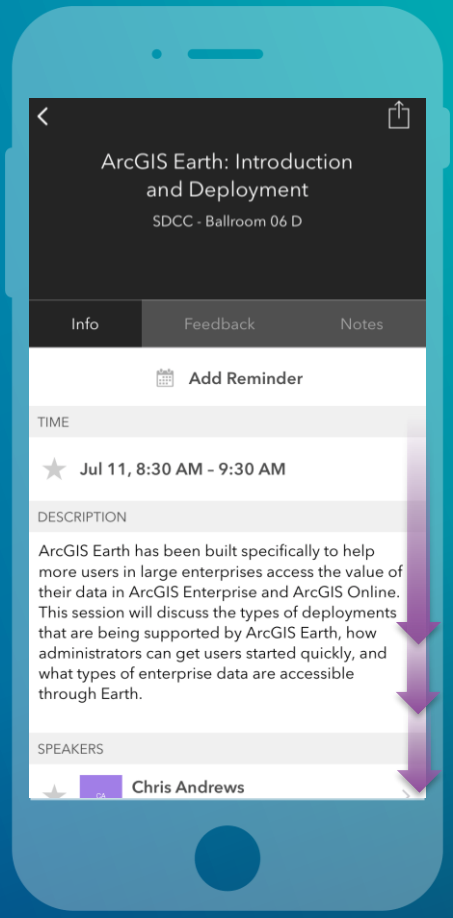
Please Take Our Survey on the App

Download the Esri Events app and find your event

Select the session you attended



Scroll down to find the feedback section



Complete answers and select “Submit”