Esri Developer Summit

March 8–11, 2016 | Palm Springs, CA

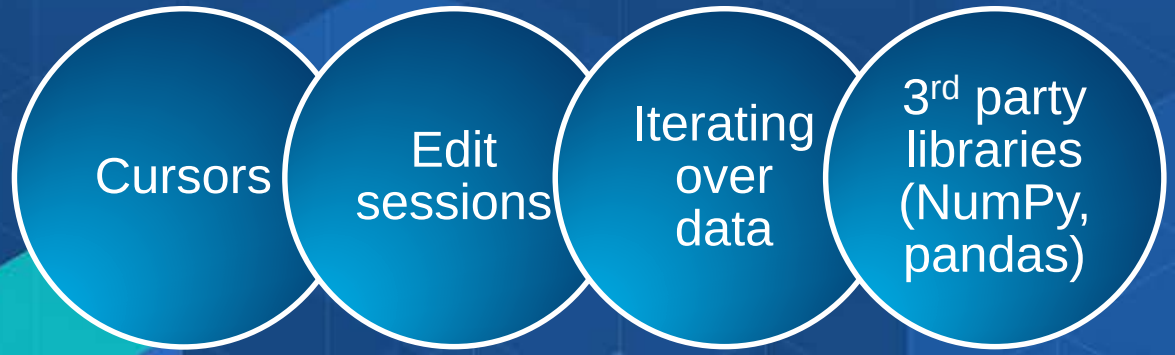
Python: Working with Feature Data

David Wynne

Ghislain Prince

Abstract

- **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 to extend analysis, managing geodatabase objects (including domains, subtypes, versions and replicas), and easily accessing data across folders and geodatabases.**



Cursors

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

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

1. **arcpy.da cursors (added at 10.1)**
2. **‘Classic’ cursors (date back to 9.0)**

- **Which one? Unless you have legacy code you don’t want to update, default to arcpy.da cursors**
  - **Superior performance**

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])
```

- **Different from ‘classic’ cursors**
  - **Work with 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 statement
  - Provide clarity
  - Other benefits: such as allowing edits on multiple tables in the same workspace

Fields and tokens

- For best performance, use only those fields you need

- Tokens can be also be used
  - Get only what you need : asking for full geometry is 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'
```

```
lookup = {u'Le\xf3n Cort\xe9s'      : u'Le\xf3n Cort\xe9s Castro',
          u'V\xe1squez de Coronado' : u'V\xe1zquez de Coronado' }


with arcpy.da.UpdateCursor("canton",
                           field_names = ("NAME_2", "Canton", "area_km2","pop_2008"),
                           where_clause = u"NAME_2 IN ('Le\xf3n Cort\xe9s', 'V\xe1squez de Coronado') ") as f
    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

```
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
  - 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'])
```

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 ArcSDE 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 aborted, and 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})
stopEditing(save_changes)
startOperation()
stopOperation()
abortOperation()
undoOperation()
redoOperation()

```
try:
    with arcpy.da.Editor(arcpy.env.workspace) as editor:
        arcpy.CalculateField_management("canton", "pop_per_km2",
                                        expression="!pop_2008! / !area_km2!",
                                        expression_type="PYTHON")
        pprint_table("canton")
        arcpy.CalculateField_management("canton", "provincia",
                                        expression="!NAME_1!",
                                        expression_type="PYTHON")
except arcpy.ExecuteError as e:
    print(e)
```

Editing with arcpy.da.Editor

esriurl.com/10618

Iterating over data

- **Many processes require cataloging or iterating over data**
- **Common theme are arcpy list functions**
  - **Over 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 vs. the hard way: http://esriurl.com/5932

```
# Using arcpy.da.ListSubtypes
from pprint import pprint
walker = arcpy.da.Walk(os.getcwd(), datatype="FeatureClass")
for dirpath, dirnames, filenames in walker:
    for 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
- RasterToNumPyArray / NumPyArrayToRaster
  - Added at 10.0 to support converting rasters to and from numpy arrays

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

```
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)
```

- Need to output 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
- Added to our stack at 10.4
- Added to Pro 1.1 (and to the external Python at 1.0)

```
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)
```

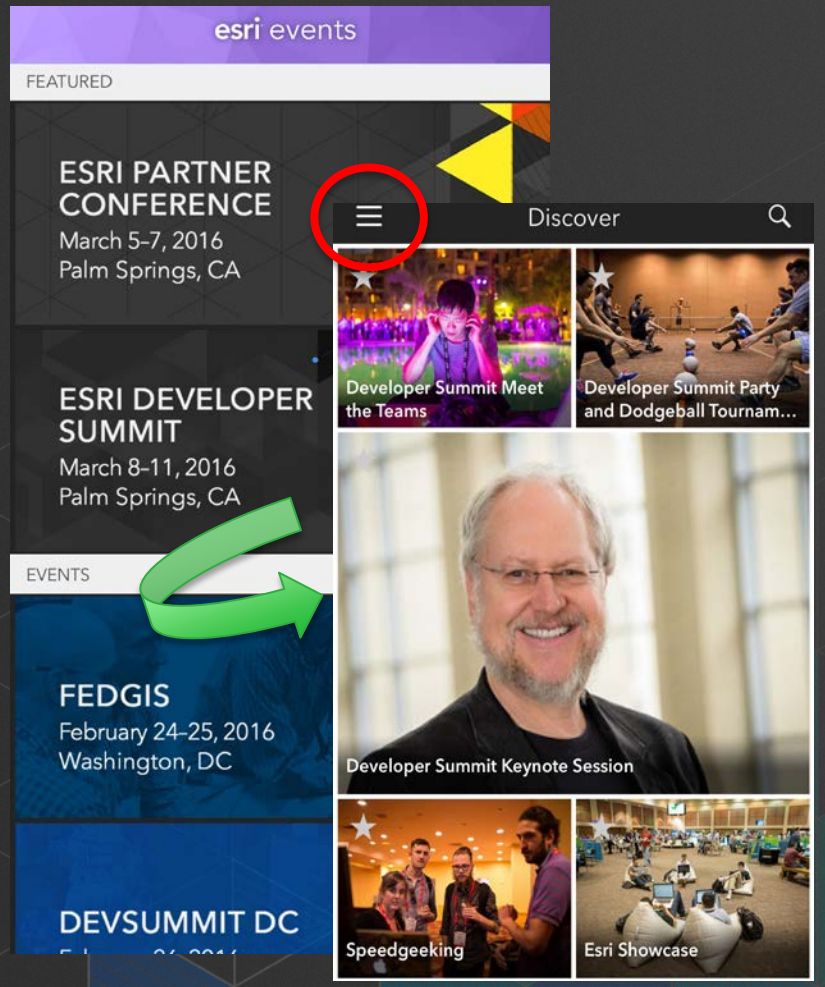
```
with arcpy.da.InsertCursor(in_fc, ['SHAPE@', id_field]) as cursor:
    unique_array = numpy.unique(in_array[id_field])  # unique ids

    # Iterate through unique sets, get array that matches unique
    # value, convert coordinates to a list and insert via cursor.
    for unique_value in unique_array:
        a = 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

return
```

arcpy, numpy and pandas

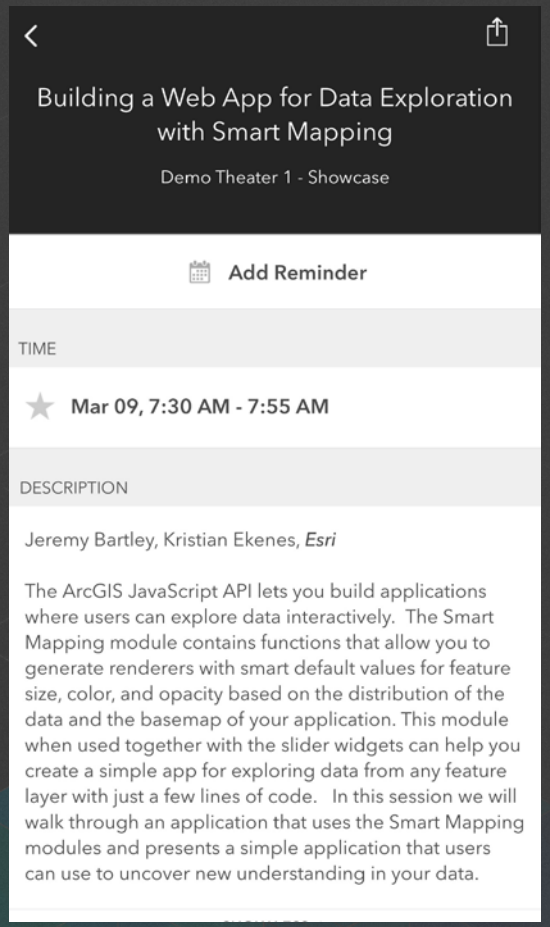
esriurl.com/10618

Please Take Our Survey! – No more memorizing Session ID numbers!! ☺

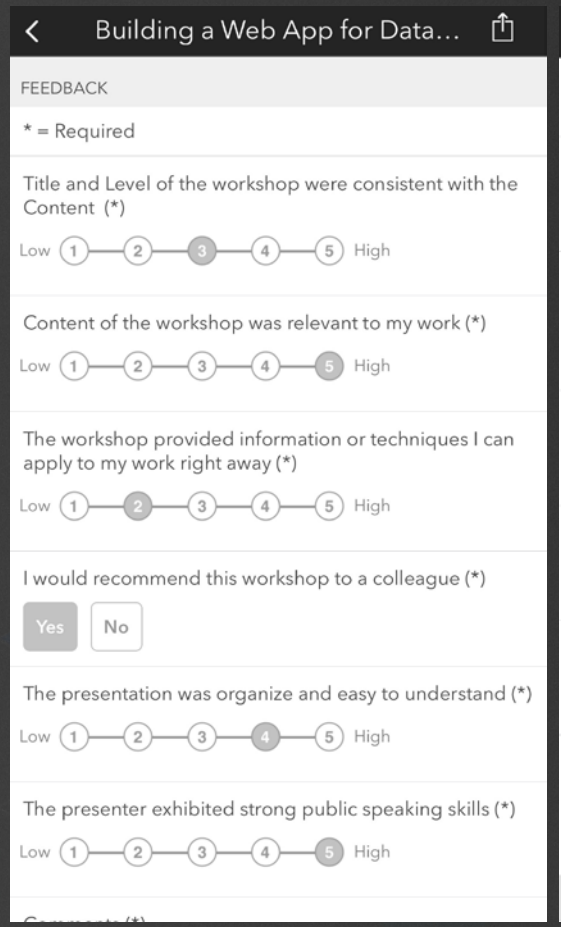
Download the Esri Events app and find your event



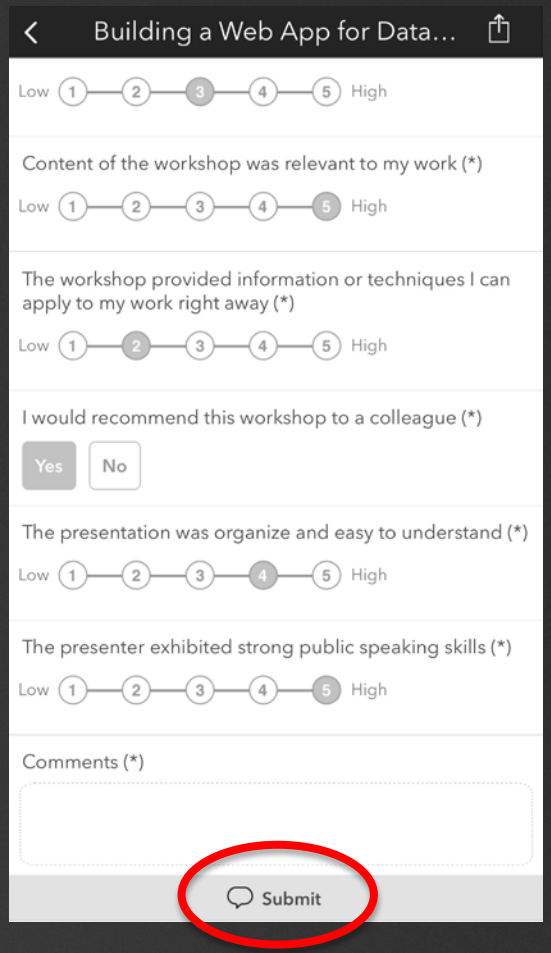
Select the session you attended



Scroll down to the "Feedback" section



Complete Answers, add a Comment, and Select "Submit"




esri®