



DEVELOPER SUMMIT

March 10–13



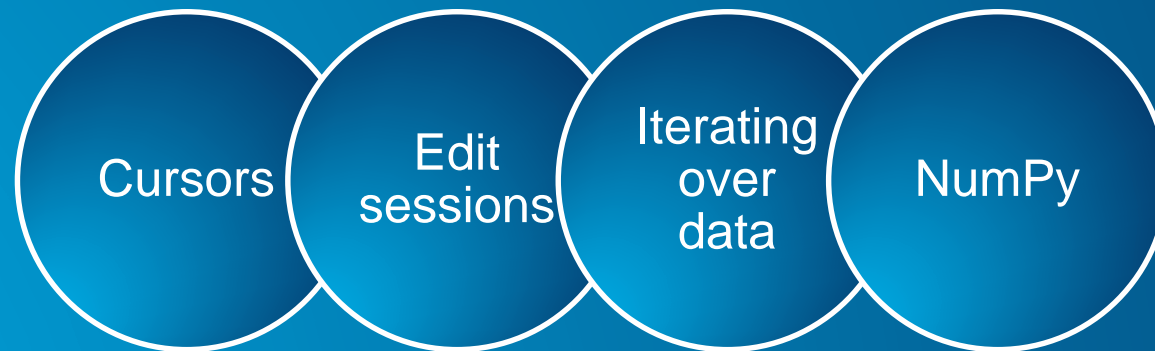
WELCOME

ArcPy: 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**

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, should default to using arcpy.da cursors**
 - **Superior performance**

SearchCursor

Read-only access

UpdateCursor

Update or delete rows

InsertCursor

Insert rows

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

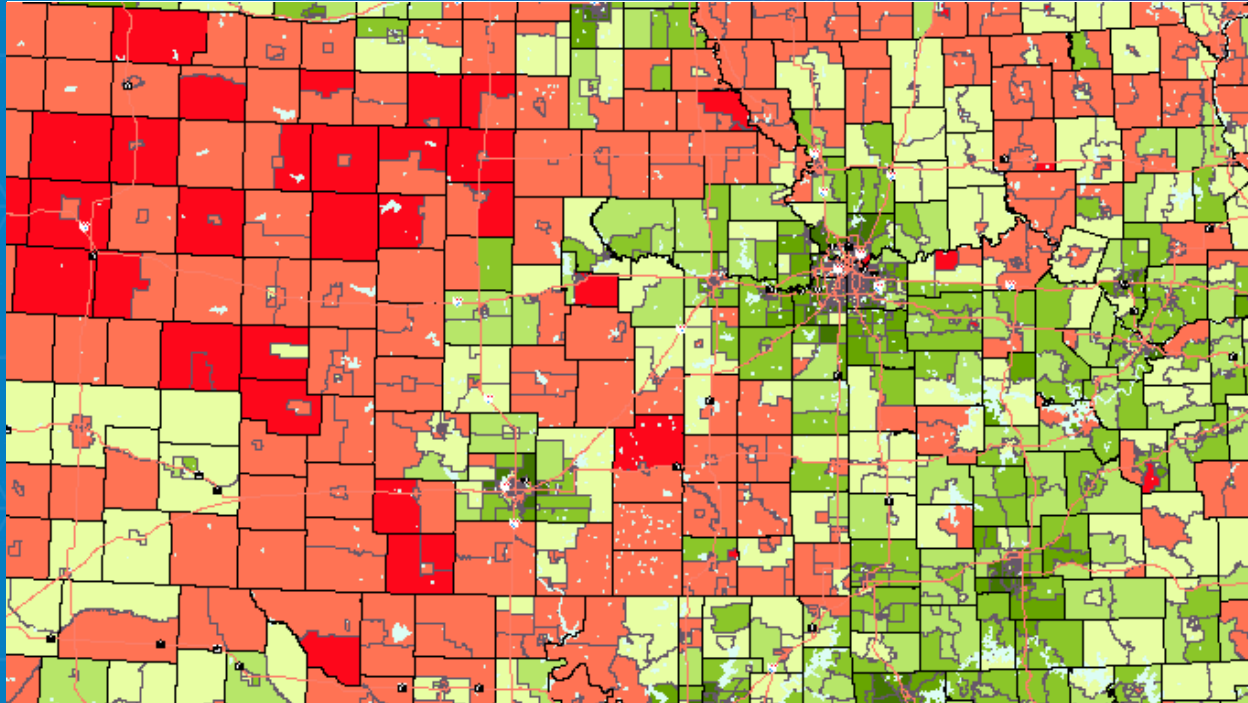
```
• with arcpy.da.SearchCursor(table, field) as cursor:  
•     for row in cursor:  
•         print row[0]
```

- **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'
```

Demo: cursors

esriurl.com/7537

Editor class

- **Supports edit sessions and edit operations**
- **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

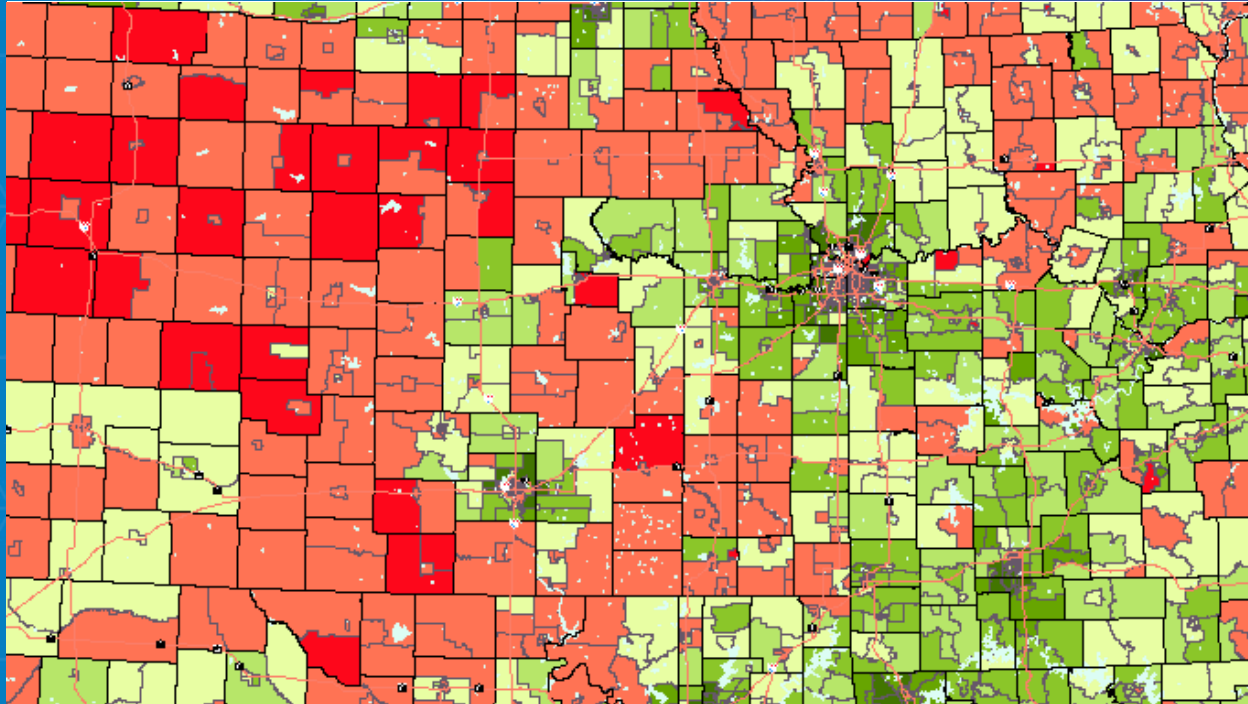
Editor class

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

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.

Editor class

```
# Start an edit session.  
• edit = arcpy.da.Editor(workspace)  
  
# Edit session is started without an undo/redo stack for versioned data  
# (for second argument, use False for unversioned data)  
• edit.startEditing(False, True)  
  
# Start an edit operation  
• edit.startOperation()  
  
# Your edits here..  
  
# Stop the edit operation.  
• edit.stopOperation()  
  
# Stop the edit session and save the changes  
• edit.stopEditing(True)
```



Demo: arcpy.da.Editor

esriurl.com/7537

Iterating over data

- More list functions to support workflows

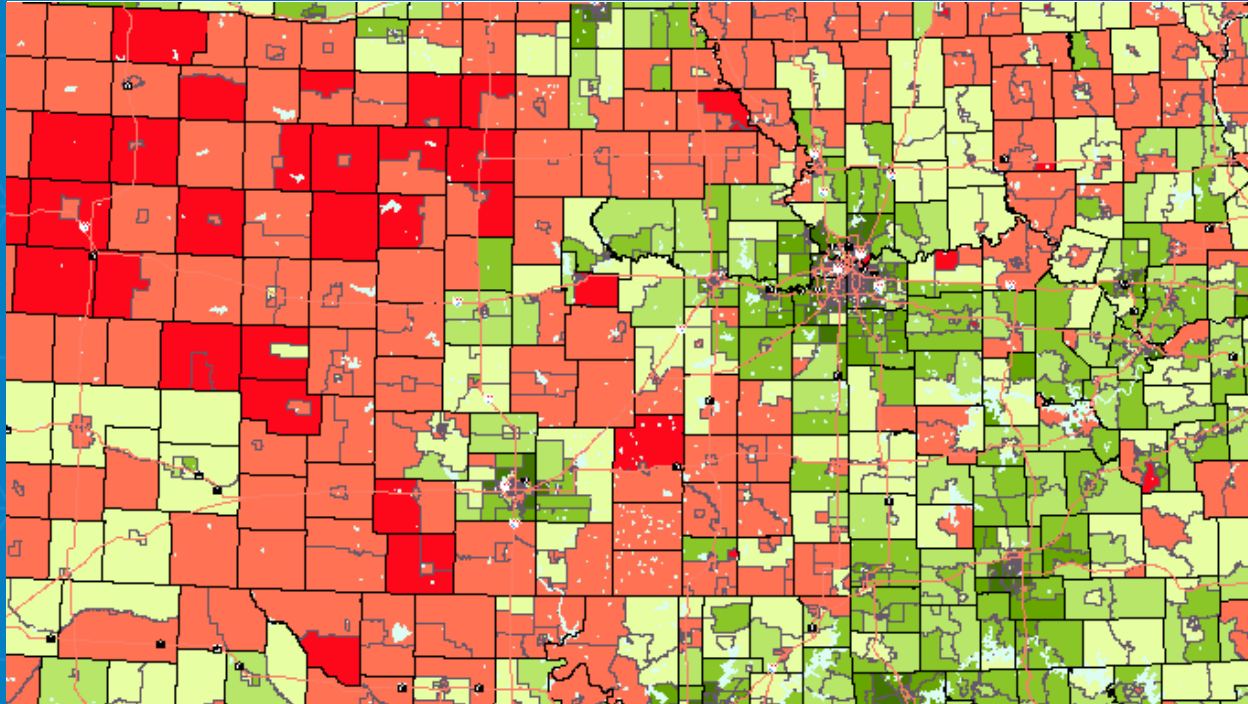
arcpy.da list functions	
ListDomains	Lists the attribute domains belonging to a geodatabase.
ListFieldConflictFilters	Lists the fields in a versioned feature class that have field conflict filters applied.
ListReplicas	Lists the replicas in a workspace.
ListSubtypes	Return a dictionary of the subtypes 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>



Demo: Iterating over data

esriurl.com/7537

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]))
```

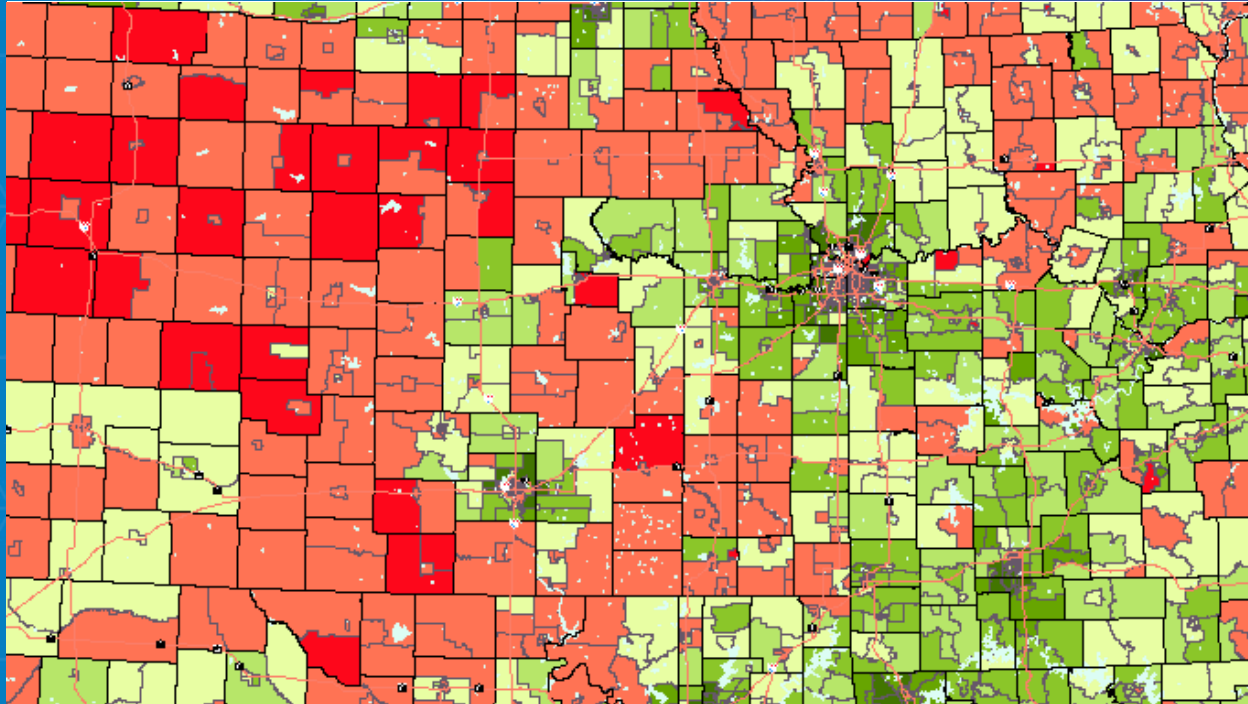
Import from NumPy

- 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>



Demo: numpy and arcpy

esriurl.com/7537

Rate This Session

www.esri.com/RateMyDevSummitSession