Python: Beyond the Basics
David Wynne, Jon Bodamer
Python: Beyond the Basics

Synopsis

This session is aimed at those with Python experience and who want to learn how to take Python further to solve analytical problems. This session will include accessing data with cursors, working with geometry, using third-party libraries, and creating Python-based geoprocessing tools.
Python: Beyond the Basics

Agenda

1. Data structures & functions
2. Creating your own tools
3. Cursors
4. Features & Geometry
5. 3rd party libraries
6. The road ahead
Functions & data structures
Defining Functions

- Organize and re-use functionality

```python
def raster_to_features(in_raster, out_features, extent=None, field=None):
    """
    Convert raster to a feature class. Output will be clipped if an extent is provided.
    """
    arcpy.env.extent = extent
    arcpy.RasterToPolygon_conversion(in_raster, out_features, extent=extent, raster_field=field)
    arcpy.ClearEnvironment('extent')
    return out_features

raster_to_features(raster1, out_features, extent=boundary)
```

- [https://docs.python.org/2/tutorial/controlflow.html#defining-functions](https://docs.python.org/2/tutorial/controlflow.html#defining-functions)
Key Python data structures

- **Lists**
  - Flexible
  - Ordered

- **Tuples**
  - Immutable
  - Ordered

- **Dictionary**
  - Key/value pairs

- [https://docs.python.org/2/tutorial/datastructures.html](https://docs.python.org/2/tutorial/datastructures.html)
List comprehension

• Compact way of mapping a list into another

```python
>>> distances = [10, 50, 200]
>>> new_distances = ['{} feet'.format(d) for d in distances]
>>> print(new_distances)
['10 feet', '50 feet', '200 feet']

>>> field_names = [f.name for f in arcpy.ListFields(table)]
>>> print(field_names)
[u'OBJECTID', u'NAME', u'ADDRESS']

>>> field_names = [f.name for f in arcpy.ListFields(table) if not f.required]
>>> print(field_names)
[u'NAME', u'ADDRESS']
```
Data structures
Creating geoprocessing tools
Geoprocessing Framework and Python

• Tools can be called from Python
• Python code can be wrapped into a tool

• Custom Python tools …
  - Looks and behaves like system tools
  - Provides default validation
  - Have no UI programming
Creating geoprocessing tools

Script tools

• A tool does 3 types of work
  1. Defines its parameters
  2. Validates its parameters
  3. Executes code that performs the actual work

Python toolboxes
Getting and setting parameters

- Parameters are received using either:
  - `arcpy.GetParameterAsText` : value is a string
  - `arcpy.GetParameter` : appropriate object type

- You can also send a parameter value back for a derived output/value with:
  - `arcpy.SetParameterAsText`
  - `arcpy.SetParameter`
Messages in a script tool

- Add custom messages into the tool’s messages
  - AddError
  - AddMessage
  - AddWarning
  - AddIDMessage

- **Note**: Error messages do not raise an Exception
- For unhandled exceptions, all exception messages are added to the tool messages
Script tools
Cursors
Cursors

- Use cursors to access records and features

<table>
<thead>
<tr>
<th>Cursor Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SearchCursor</td>
<td>Read-only</td>
</tr>
<tr>
<td>UpdateCursor</td>
<td>Update or delete rows</td>
</tr>
<tr>
<td>InsertCursor</td>
<td>Insert rows</td>
</tr>
</tbody>
</table>

- Two varieties
  - ‘Data access’ cursors (10.1 onwards)
  - ‘Classic’ cursors
Cursor mechanics

• Data access cursors use lists and tuples
  - Values are accessed by index

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

• Classic cursors use row objects
  - Values are accessed by setValue/getValue

```python
    cursor = arcpy.InsertCursor(table)  
    row = cursor.newRow()  
    row.setValue("field1", 1)  
    row.setValue("field2", 10)  
    cursor.insertRow(row)
```
Cursor performance

• Use only those fields you need

• Use tokens
  - Get only what you need
  - Full geometry is expensive
Cursors
Geometry
Geometry and cursors

- Can create geometry in different ways
  - Geometry objects
  - List of coordinates
  - Using other formats
    - JSON, WKT, WKB
Working with geometry

- Relational:
  - Is a point within a polygon?

```
point.within(polygon)
```
Working with geometry

• Topological
  - What is the intersection of two geometries?

```python
poly1.intersect(poly2)
```
Working with geometry

• More:
  - What is the halfway point of a line?
    ```python
    line.positionAlongLine(0.5, True)
    ```
  - What is the geodesic area of a polygon?
    ```python
    poly.getArea('GEODESIC')
    ```
Geometry
3rd party libraries
3rd party libraries

- Python has a rich set of 3rd party libraries
  - [https://pypi.python.org/pypi](https://pypi.python.org/pypi)

- We include several to support tools and other functionality

- NumPy
  - A powerful array object
  - Sophisticated analysis capabilities
  - arcpy support conversion to and from rasters, feature classes, and tables
3rd party library
- Request
Road ahead
Road ahead

• arcpy is supported in ArcGIS Pro
  - arcpy.mapping has evolved
  - A subset of geoprocessing tools will disappear

• ArcGIS Pro will use Python 3.4
  - Your Python code *may* be okay as is
  - Definitely possible to write code that will work in both Python 2 and 3
Planning ahead

• Resources:
  - Python’s 2to3 utility
  - Analyze Tools For Pro
  - python3porting.com
Thank you...

- Please fill out the session survey:

  Tuesday Offering ID:  1114
  Wednesday Offering ID:  1243
  Friday Offering ID:  1997

  Online – [www.esri.com/ucsessionsurveys](http://www.esri.com/ucsessionsurveys)
  Paper – pick up and put in drop box