



# Python: Beyond the Basics

David Wynne, Jon Bodamer

Technical Workshop

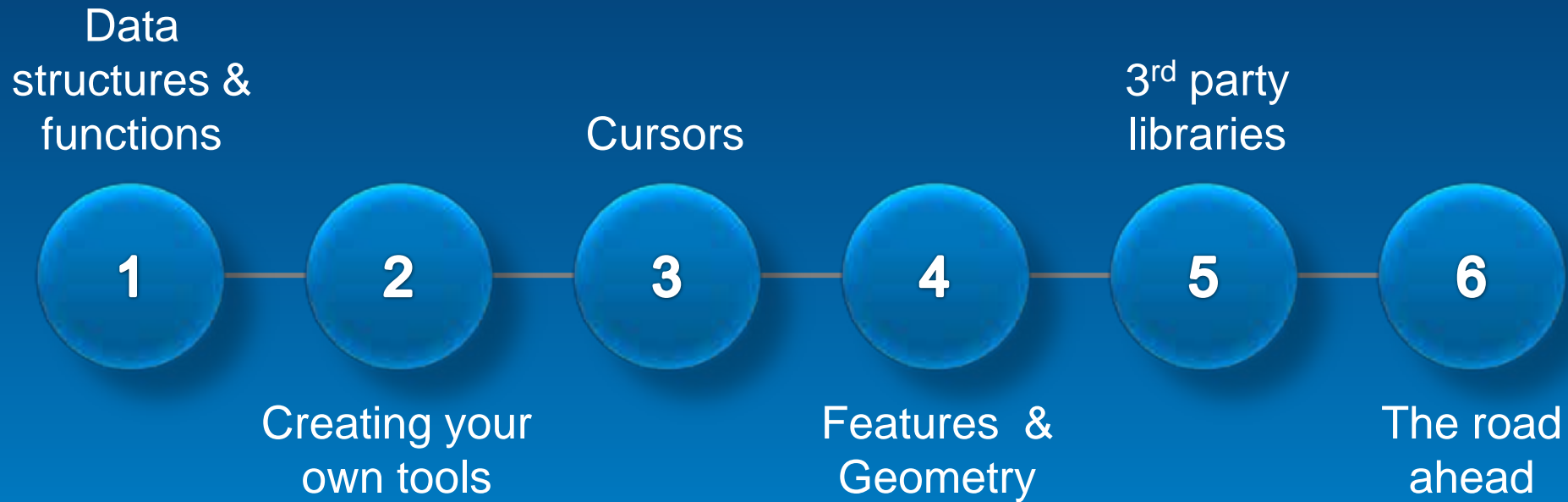
# Python: Beyond the Basics

## 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



# Functions & data structures

# Defining Functions

- Organize and re-use functionality

```
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,  
                                    raster_field=field)  
    arcpy.ClearEnvironment('extent')  
    return out_features  
  
raster_to_features(raster1, out_features, extent=boundary)
```

Define your  
function

Return a  
result  
Call the  
function

- <https://docs.python.org/2/tutorial/controlflow.html#defining-functions>

# Key Python data structures

- **Lists**

- Flexible
- Ordered

- **Tuples**

- Immutable
- Ordered

- **Dictionary**

- Key/value pairs

```
• l = ['10 feet', '20 feet', '50 feet']
```

```
• t = ('Thurston', 'Pierce', 'King')
```

```
• d = {'ProductName': 'desktop',  
      'InstallDir': 'c:\\ArcGIS\\Desktop10.1'}
```

- <https://docs.python.org/2/tutorial/datastructures.html>

# List comprehension

- Compact way of mapping a list into another

```
• >>> 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']
```

Demo

# Data structures



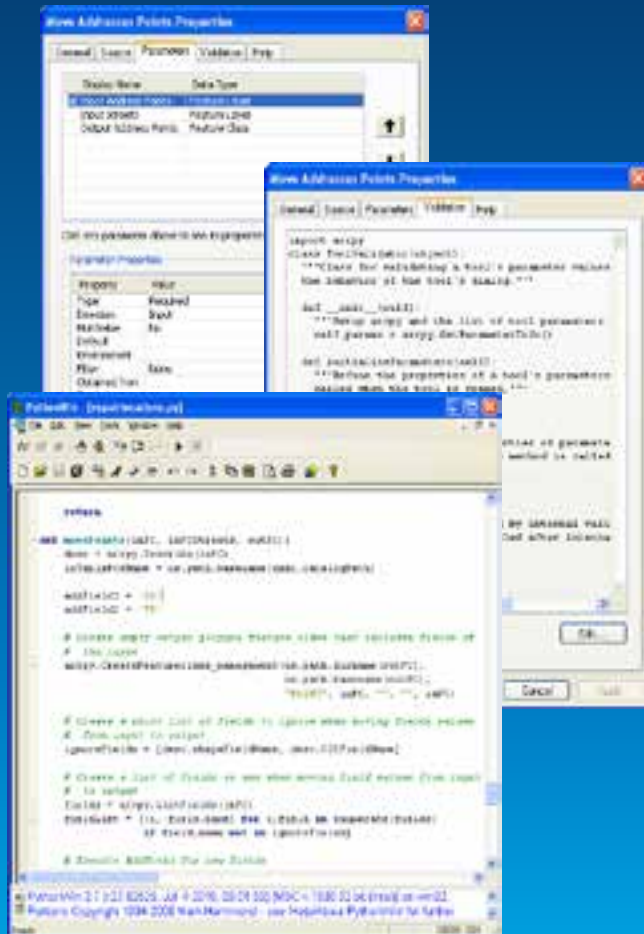
# Creating geoprocessing tools

# 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

```
class Tool(object):
    def __init__(self):
        """Define the tool (tool name is the name of the class)"""
        self.label = "Tool"
        self.description = ""
        self.canRunInBackground = False

    def getParameterInfo(self):
        """Define parameter definitions"""
        params = None
        return params

    def isLicensed(self):
        """Set whether tool is licensed to execute."""
        return True

    def updateParameters(self, parameters):
        """Modify the values and properties of parameters
        validation is performed. This method is called
        whenever a parameter has been changed."""
        return

    def updateMessages(self, parameters):
        """Modify the messages created by internal tool
        parameter. This method is called after internal
        parameter validation has finished."""
        return

    def execute(self, parameters, messages):
        """The source code of the tool."""
        return
```

# 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**

Demo

# Script tools

# Cursors

# Cursors

- Use cursors to access records and features

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

- Two varieties
  - 'Data access' cursors (10.1 onwards)
  - 'Classic' cursors



# Cursor mechanics

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

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

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

```
• 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*

- **SHAPE@XY** —A tuple of the feature's centroid x,y coordinates.
- SHAPE@**SHAPE@XY** —A tuple of the feature's true centroid x,y coordinates.
- SHAPE@X —A double of the feature's x-coordinate.
- SHAPE@Y —A double of the feature's y-coordinate.
- SHAPE@Z —A double of the feature's z-coordinate.
- SHAPE@M —A double of the feature's m-value.
- SHAPE@JSON — The esri JSON string representing the geometry.
- SHAPE@WKB —The well-known binary (WKB) representation for OGC geometry. It provides a portable representation of a geometry value as a contiguous stream of bytes.
- SHAPE@WKT —The well-known text (WKT) representation for OGC geometry. It provides a portable representation of a geometry value as a text string.
- SHAPE@ —A [geometry](#) object for the feature.
- SHAPE@AREA —A double of the feature's area.
- SHAPE@LENGTH —A double of the feature's length.
- OID@ —The value of the [ObjectID](#) field.

Demo

# Cursors

# Geometry

# Geometry and cursors

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

```
• line = arcpy.Polyline(  
•     arcpy.Array([arcpy.Point(2, 3),  
•                 arcpy.Point(3, 5)])  
• line = [(2, 3), (3, 5)]  
• line = '{"paths": [[[2, 3], [3, 5]]]}'  
• cursor.insertRow([line])
```

# Working with geometry

- Relational:
  - Is a point within a polygon?

```
• point.within(polygon)
```

```
boundary  
buffer  
clip  
contains  
convexHull  
crosses  
difference  
disjoint  
distanceTo  
equals  
getArea  
getLength  
getPart  
intersect  
overlaps  
positionAlongLine  
projectAs  
symmetricDifference  
touches  
union  
within
```

# Working with geometry

- Topological
  - What is the intersection of two geometries?

```
poly1.intersect(poly2)
```

```
boundary  
buffer  
clip  
contains  
convexHull  
crosses  
difference  
disjoint  
distanceTo  
equals  
getArea  
getLength  
getPart  
intersect  
overlaps  
positionAlongLine  
projectAs  
symmetricDifference  
touches  
union  
within
```

# Working with geometry

- More:

- What is the halfway point of a line?

```
• line.positionAlongLine(0.5, True)
```

- What is the geodesic area of a polygon?

```
• poly.getArea('GEODESIC')
```

```
boundary  
buffer  
clip  
contains  
convexHull  
crosses  
difference  
disjoint  
distanceTo  
equals  
getArea  
getLength  
getPart  
intersect  
overlaps  
positionAlongLine  
projectAs  
symmetricDifference  
touches  
union  
within
```



Demo

# Geometry

# 3<sup>rd</sup> party libraries

## 3<sup>rd</sup> party libraries

- **Python has a rich set of 3<sup>rd</sup> party libraries**
  - <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**

Demo

# 3<sup>rd</sup> 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](http://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**





Understanding our world.