Extending ArcGIS with Python

USGS GIS Workshop, May 10-11th, 2011

Drew Flater - Esri

Does this describe you?

- Comfortable using ArcGIS, geoprocessing, but want to become more efficient
- Some experience with Python, or other language
- Want to pick up some tips & tricks
- Ask the question *What else?* – extend ArcGIS to do things that are not “in the box”

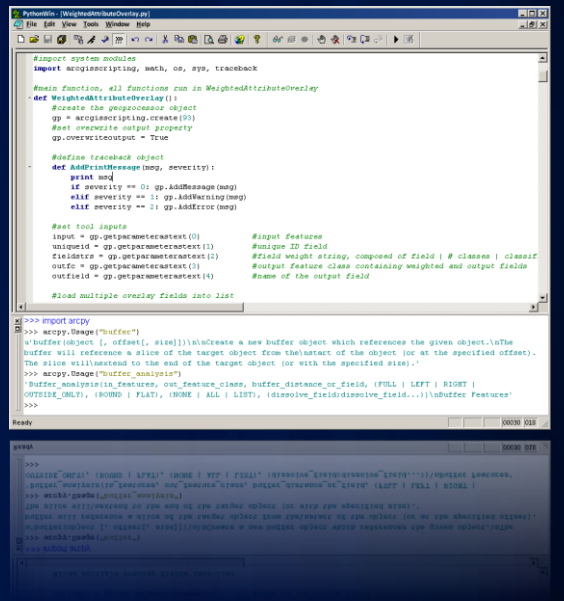
Agenda

- **Python scripting essentials**
  - Why use Python scripting?
  - Python 101
  - What is ArcPy?
  - Executing geoprocessing tools
  - Creating workflows
- **Batch Processing**
  - *Listing* data
- **Reading and Creating Data**
  - Describe
  - Cursors
- **Spatial Analyst module**
  - Functions
  - Map algebra
  - Raster classes
- **Map Automation**
  - ArcPy Mapping module
  - Repair broken data sources
  - Create map books
- **Creating script tools**

Python Scripting Essentials

Why use Python scripting?

- Scripting language of ArcGIS
- Free, cross-platform, easy to learn, established community
- But why? Other ways to run tools
- Develop, execute, and share geoprocessing workflows
- Improve productivity



Python 101

- Where do I write Python code?
  - IDE like PythonWin; Python window in ArcGIS
- Which lines will run?

```
# I am a comment, I will not execute
import arcpy
```

- What are variables?
  - A name that stores a value; assigned using =

```
input = "C:/Data/roads.shp"
distance = 50
both = ["C:/Data/roads.shp", 50]

# Variables act as substitutes for raw values
arcpy.Buffer_analysis(input, "output.shp", distance)
```

Python 101

- Python has logic for testing conditions
  - if, else statement
  - Colon at end of each condition
  - Indentation determines what is executed
  - == tests equality; other operators like >, <, !=

```
var = "a"
if var == "a":
    # Execute indented lines
    print "variable is a"
else:
    print "variable is not a"
```

Python 101

- Techniques for iterating or looping
  - While loops, counted loops, list loops
  - Colon at end of statement
  - Indentation determines what is executed

```
x = 1
while x < 5:
  print x
  x = x + 1

for num in range(1,5):
  print num

x = [1, 2, 3, 4]
for num in x:
  print num
```

Python 101

- Case sensitivity
  - Variables, functions, etc. are case sensitive
  - name ‘X’ is not defined, function ‘X’ does not exist
- For paths, use forward-slash as separator

  `"C:/Data/Roads.shp"`
- Functions & Modules
  - ***Function***: a defined piece of functionality that performs a specific task; requires arguments ()
  - ***Module***: a Python file where functions live; imported
  - **math.sqrt(100) … 10.0**
  - “There’s a *module* for that!”

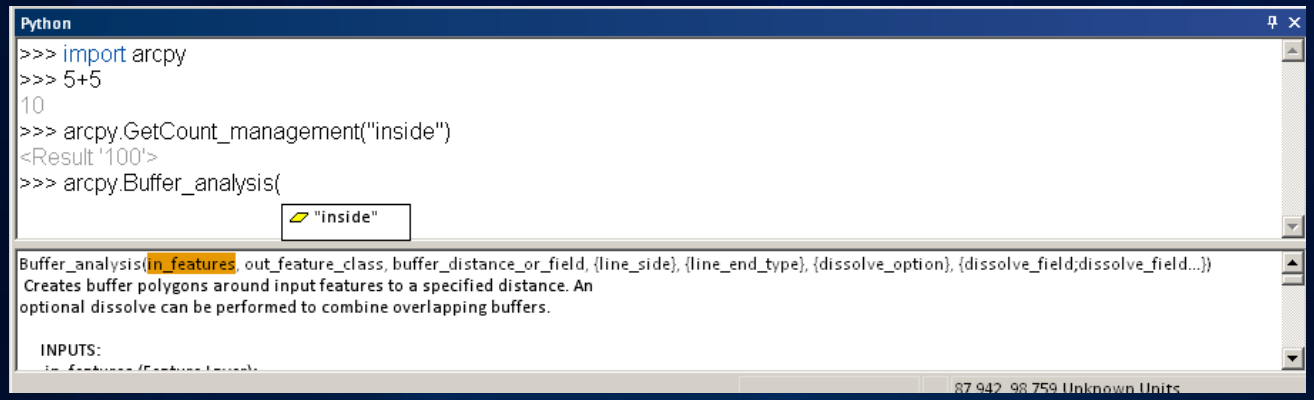
ArcPy

- The access point to geoprocessing tools
- A package of functions, classes and modules, all related to scripting in ArcGIS
  - Functions that enhance geoprocessing workflows (ListFeatureClasses, Describe, SearchCursor)
  - Classes that can be used to create complex objects (SpatialReference , FieldMap objects)
  - Modules that provide additional functionality (Mapping ,SpatialAnalyst modules)
- Builds on arcgisscripting module (pre-10.0)

ArcGIS Python window

- **Embedded, interactive Python window within ArcGIS**
  - **Access to ArcPy, any Python functionality**

- **Great for experimenting with Python and learning tool syntax**



Executing a tool in Python

- ArcPy must be imported
- Follow syntax: arcpy.toolname_toolboxalias()
- Enter input and output parameters

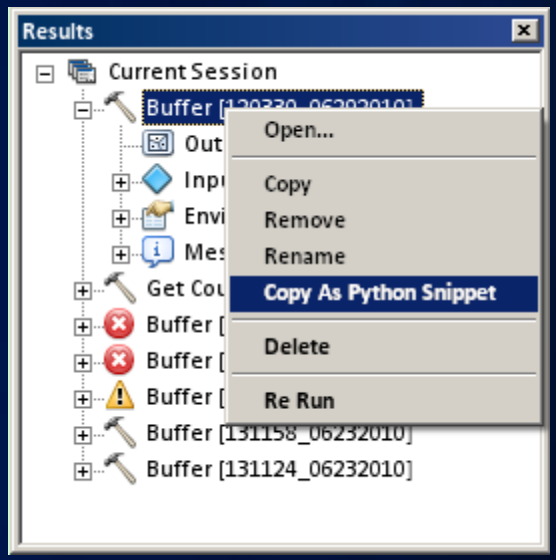
```
# Import ArcPy
import arcpy

# Set workspace environment
arcpy.env.workspace = "C:/Data"

# Execute Geoprocessing tool
arcpy.Buffer_analysis("Roads.shp", "Roads_buffer.shp",
  "50 Meters")
```

Getting tool syntax

- Results window, ‘Copy as Python Snippet’
- Export Model to Python script
- Drag tool into Python window
- Tool documentation
- arcpy.Usage(“Buffer_analysis”)



Setting environments in Python

- Accessed from arcpy.env
- Provides finer control of tool execution; makes scripting easier
- Common environments:
  - Workspace, coordinate system, extent

```
arcpy.env.workspace = "C:/Data"
arcpy.env.extent = "0 0 100 100"
```

Scripting Geoprocessing Tools

Exercise 1

10 min

Tips & Tricks

- Use the **result** object
  - Returned by geoprocessing tools
  - Maintains messages, parameters, and output

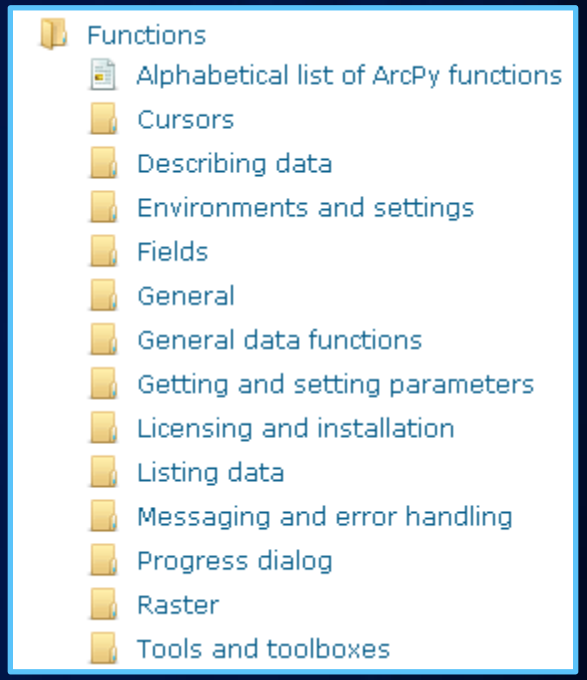
```
# Use Select and Buffer tools
select = arcpy.Select_analysis(veg, "select")
buffer = arcpy.Buffer_analysis(roads, "buffer", "75 Feet")

# Erase the output of Buffer from the output of Select
erase = arcpy.Erase_analysis(select, buffer, "erase")
```

Batch Processing

ArcPy functions

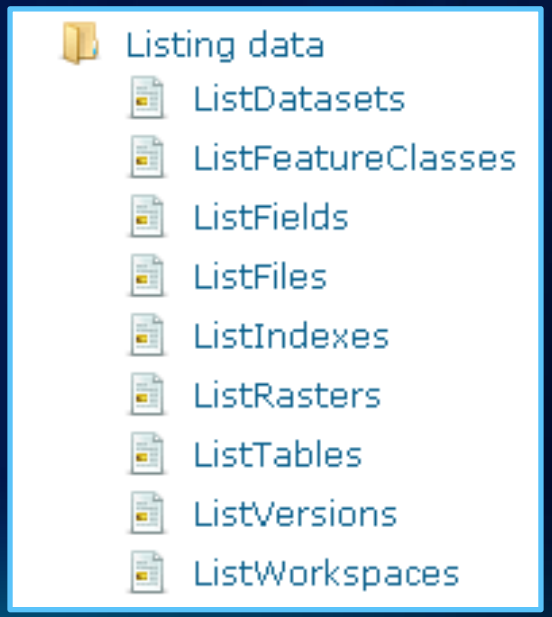
- **Perform useful scripting tasks**
  - **List data to aid batch processing (ListFeatureClasses, 12 total List functions)**
  - **Getting data properties (Describe)**
  - **Etc.**
- **Supports automation of manual tasks**



Batch processing

- **Run a geoprocessing operation multiple times with some automation**
  - **Example: Using the Clip tool to clip every feature class in a workspace to a boundary**

- **List functions used in Python to perform batch processing**



ListFeatureClasses

```
# Set the workspace
arcpy.env.workspace = "C:/Data/FileGDB.gdb/FDs"

# Get a list of all feature classes
fcList = arcpy.ListFeatureClasses()

# Print the list of feature classes one at a time
for fc in fcList:
  print fc
```

Batch Processing
Exercise 2

5 min

Tips & Tricks

- Use **string formatting** in Python to easily combine strings with other strings (or other types)
  - Placeholder using %
  - %s : convert item to string, then replace

```
# Clip each feature class, name it according to the input
for fc in fclist:
    arcpy.Clip_analysis(fc, "StudyArea",
        "Clipped/%s_clip" % fc)
```

Reading and Creating Data

Reading Data Properties

- **Describe function**
  - **Returns an object with properties**

- **Allows script to determine properties of data**
  - **Data type (shapefile, coverage, network dataset, etc.)**
  - **Shape type (point, line, polygon)**
  - **Shape field name**
  - **Etc.**

```
# Describe a feature class
desc = arcpy.Describe("C:/Data/roads.shp")

print desc.shapeType
>>> "Polyline"
```

Reading Data Values and Geometry

- **ArcPy cursors used to access table records**
  - **Iterate through each record**
  - **Retrieve field values of tables, feature classes, rasters**
  - **Get geometry of feature class features**

Cursor	Explanation
SearchCursor	Read-only access to field values, geometry
UpdateCursor	Update or delete field values, geometry (write-access)
InsertCursor	Add new records to a table or feature class; write field values and geometry

Cursors

- **SearchCursor**

```
scur = arcpy.SearchCursor("C:/Data/Roads.shp")
for row in scur:
    print row.getValue("Name")
```

- **UpdateCursor**

```
ucur = arcpy.UpdateCursor("C:/Data/Sites.shp")
for row in ucur:
    log = math.log(row.getValue("Num"))
    row.setValue("Log", log)
    ucur.updateRow(row)
```

Reading Geometry

- Feature classes have a geometry field
  - Typically *(but not always)* named **Shape**
- Returns a geometry object
  - Has properties that describe the feature
  - area, length, isMultipart, partCount, pointCount, type, ...
- Geometry objects can often be used in place of feature classes

```
# Buffer each feature to a new feature class
for row in arcpy.SearchCursor("C:/data/Roads.shp"):
    feature = row.getValue("Shape")
    name = row.getValue("Name")
    arcpy.Buffer_analysis(feature, "buffer_%s" % name,
        "100 Feet")
```

Cursors (creating new data)

- **InsertCursor used to add new rows, features**
  - **Insert values into attribute fields**
  - **Insert geometries into shape field**

```
icur = arcpy.InsertCursor("C:/Data/Cities.shp")
row = icur.newRow()
row.setValue("City", "Denver")
row.setValue("Shape", arcpy.Point(-104.98, 39.74))
icur.insertRow(row)
```

ArcPy Cursors
Exercise 3

10 min

Tips & Tricks

- Clean up cursors using a **try, except, finally** statement
  - Cursors can keep a lock on data

```
scur = arcpy.SearchCursor("C:/Data/Roads.shp")
try:
    for row in scur:
        print row.getValue("Name")
except:
    raise
finally:
    if scur:
        del scur
```

Spatial Analyst Module

Spatial Analyst Module

- `from arcpy.sa import *`
- **Includes all Spatial Analyst tools**
- **Integrates Map Algebra into Python**
  - ***Defines geographic analysis as algebraic expressions***
  - **Supports mathematical, relational, other operators**
  - **Output on the left-side**
- **Helper classes that can be used to support complex parameter**

```
demm = Raster("DEM") / 3.28
slpdeg = Slope(demm, "DEGREE")

demfs = FocalStatistics(demm, NbrRectangle(3,3), "MEAN")
```

Raster Class

- Used as input to tools and map algebra expressions
- Reference to raster on disk, created in two ways:
  - Returned output from Spatial Analyst functions
  - Cast using Raster() function
- Creates a temporary raster dataset that must be saved to be made permanent
- Has properties and method
  - raster.minimum, raster.format, raster.extent, etc.
  - raster.save()

Raster Integration

- NumPy is a 3rd party Python library for numeric or mathematical computing
  - A powerful array object
  - Sophisticated analysis capabilities
  - NumPy arrays used in other packages like SciPy

- Rasters can be converted to NumPy arrays
  - RasterToNumPyArray(), NumPyArrayToRaster()

Raster to NumPy Array

- **Example: Correlation Coefficient between two Rasters**

```
# Find the correlation between Temperature and
  Barometric Pressure

# Convert Temperature and Pressure rasters to arrays
tempArray = arcpy.RasterToNumPyArray("Temperature")
presArray = arcpy.RasterToNumPyArray("Baro_Pressure")

# Flatten the arrays
tempArray.ravel()
presArray.ravel()

# Print the correlation matrix
print numpy.corrcoef(tempArray, presArray)
>>> [[1.0   0.98]
     [0.98  1.0 ]]

# There is a 0.98 correlation coefficient (R)
```

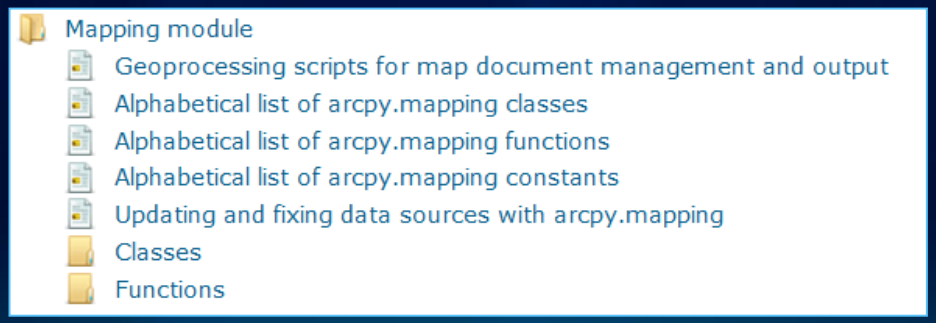
Spatial Analyst module
Exercise 4

10 min

ArcPy Mapping module

ArcPy Mapping module

- **Module that contains functions, classes used to automate mapping tasks**
  - **Manage map documents, layer files, and the data within**
  - **Find and fix broken data sources**
  - **Update a layer's symbology across many MXDs**
  - **Export and print map documents**
  - **Map production/map series**



ArcPy Mapping module

- **MapDocument object is essential**
  - **References mxd on disk; has methods and properties**
- **Needed to perform most mapping tasks**
  - **MapDocument as input to function**

OR

  - **Functions called from MapDocument**

```
md = arcpy.mapping.MapDocument("C:/Maps/NtlParks.mxd")

# Set map document properties
md.description = "Map of National Parks"
# List layers in the map
maplayers = arcpy.mapping.ListLayers(md)
# Fix Data Sources
md.replaceWorkspaces(….)
# Export the map to PDF
arcpy.mapping.ExportToPDF(md, "C:/Maps/NtlParks.pdf")
```

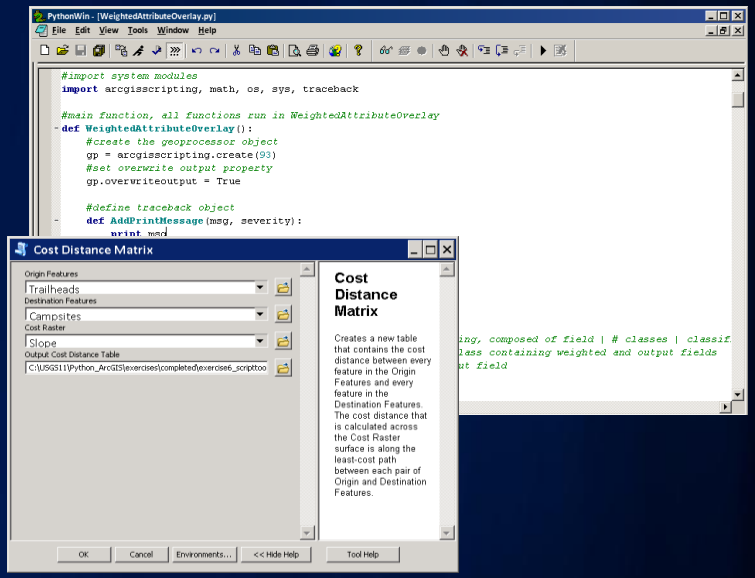
ArcPy Mapping module
Exercise 5

10 min

ArcGIS Script Tools

Script Tools

- **Script tools are the best way to create and share custom geoprocessing functionality**
  - More people know how to run a tool than a Python script
- Source is a script
- It is a tool
  - Use in ModelBuilder, scripts
  - Has inputs, outputs; generic
- Communicates with application
  - Layers added to map
  - Messages
- Vehicle to serve geoprocessing tasks through ArcGIS Server



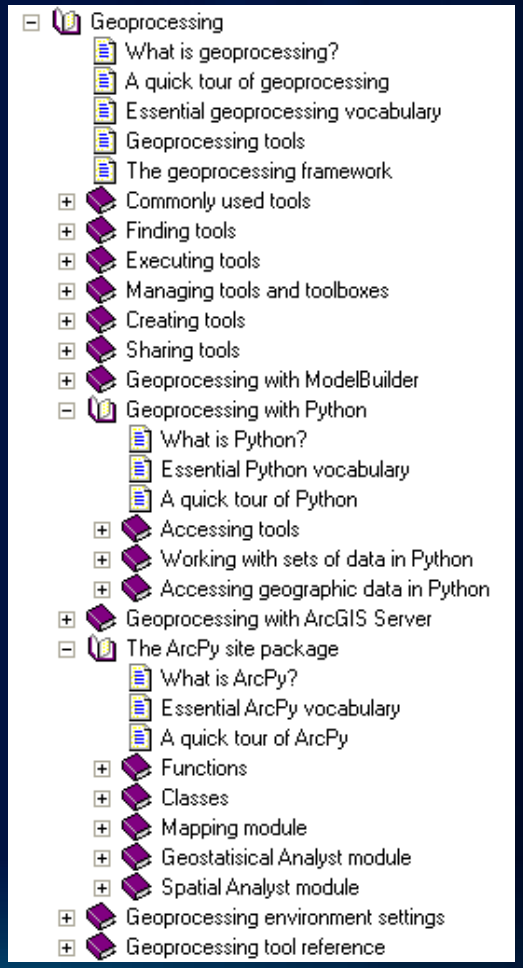
Creating an ArcGIS Script Tool

Exercise 6

5 min

Python Scripting Resources

- ArcGIS Resource Centers
  - resources.arcgis.com
  - Online documentation
  - Geoprocessing Resource Center: script gallery, blog, presentations
- Python References
  - *Learning Python* by Mark Lutz
  - *Core Python Programming* by Wesley J. Chun
- Python Organization
  - python.org



Esri Training for Python

esri.com/training

- **Instructor-Led Course**
  - **Introduction to Geoprocessing Scripts Using Python**
- **Web Course**
  - **Using Python in ArcGIS Desktop 10**