Week 10 Lecture:
Data and Batch Geoprocessing With ArcGIS

Introduction to Programming for GIS & Remote Sensing
GEO6938-1469
GEO4938-147A
Re-cap From Past Lectures

• We’ve discussed modularization
  – Subroutines
  – Functions
  – Object-orientation

• All this leads to one thing:
  – Code Re-use
    • Functions
    • Imports
    • Packages
Code Re-Use

• The goal is to make life easier for us!

• Complex operations and procedures can be stored
  – Including documentation ➔ Straight into articles...
  – Write once... run many times
  – Entire processing frameworks can be incorporated:
    • ArcGIS’ Geoprocessing!
Objectives This Week

• Learn about ArcGIS’ Geoprocessing objects

• Python facilities within ArcGIS

• Building Models and integrating with scripts

• Interface ArcGIS and other tools using Python as “glue”
What is the ArcGIS Geoprocessing Framework?

• An “ArcObjects” collection
  – Really just a single object with many, many sub-components

• Allows accessing geoprocessing tools as native methods
From Last Week: Models/Tools to Python
# -*- coding: utf-8 -*-

# data_import.py
# Created on: 2013-10-14 14:52:18.00000
# (generated by ArcGIS/ModelBuilder)
# Description:
#
#
# Import arcpy module
import arcpy

# Local variables:
tile_one_txt = "H:\\tmp\\tmp2\\output\\tile-one.txt"
tile_one_tif = "H:\\tmp\\tmp2\\output\\tile-one.tif"

# Process: ASCII to Raster
arcpy.ASCIIToRaster_conversion(tile_one_txt, tile_one_tif, "INTEGER")
Pre-ArcGIS 10.0: The Geoprocessor Object

• How do we instantiate the Geoprocessor?

```python
import arcgisscripting
gp = arcgisscripting.create()
```

• How do we interact with the Geoprocessor?

```python
gp.clip_analysis(In,Clip,Out)  # alias for:
gp.Toolbox = "Analysis"
gp.clip(In,Clip,Out)
```
Post-ArcGIS 10.0: The arcpy Object

• How do we instantiate the Geoprocessor?
  
  ```python
  import arcpy
  ```

• How do we interact with the Geoprocessor?
  
  ```python
  arcpy.Clip_management( ... )
  arcpy.CheckOutExtension("Spatial")
  from arcpy.sa import *
  ```

  ```python
  arcpy.env.overwriteOutput = overwrite
  ...  ```
The arcpy Geoprocessor Object

- Uses environment settings to define the operating conditions, input/output, defaults
So Geoprocessing Opens Doors

• Besides access to ArcToolbox tools it gives properties and methods to create and manipulate datasets directly!

• Explore the object diagram to find functionality...
  – Cataloguing
  – Describing
  – Listing
  – Editing
The Geoprocessor Acts as Gateway...

• To all of the power of ArcGIS and its tools:

  Geoprocessor Programming Model
This version of the Geoprocessor Programming Model shows the methods, properties, and objects as supported when creating the geoprocessor using the import arcpy command.

```python
import arcpy
```

**arcpy**

- MaxSeverity
- MessageCount
- OverwriteOutput: Boolean
- ParameterCount
- ScriptVersion
- Toolbox

- AddError (Message)
- AddFieldDelimiters (FieldName, Workspace)
- AddIDMessage (Type, ID, Argument1, Argument2)
- AddMessage (Message)
- AddReturnMessage (Index)
- AddToolbox (Toolbox)
- AddWarning (Message)
- ClearEnvironment (Environment)
- Command (CommandLineString)
- CopyParameter (fromIndex, toIndex)
- CheckExtension (ExtensionCode)
- CheckInExtension (ExtensionCode)
- CheckOutExtension (ExtensionCode)
- CheckProduct (ProductCode)

- CreateObject (ObjectName, Argument1, Argument2, Argument3, Argument4, Argument5): Object

**Projected Coordinate System only**

- CentralMeridian
- CentralMeridianInDegrees
- LongitudeOfOrigin
- LatitudeOf1st
- LatitudeOf2nd
- FalseEasting
- FalseNorthing
- CentralParallel
- StandardParallel1
- StandardParallel2
- LongitudeOf1st
- LongitudeOf2nd
- ScaleFactor
- Azimuth
- Classification
- PCSName
- PCSCode
- ProjectionName
- ProjectionCode
- LinearUnitName
- LinearUnitCode
Putting it all together…

• Python gives you full functionality to programming environment
  – Operating/file system integration
  – Fully extensible
  – All the logic you can stomach

• Geoprocessing framework gives you all the tools in ArcToolbox plus ability to create and edit data directly

... == “Amazing Possibilities”
Cataloguing and Listing Spatial Data

- One common and critical task:
  - Batch Processing

- We do this with arcpy’s List methods
  - Use for different data types
  - List objects for iteration

<table>
<thead>
<tr>
<th>ListMethods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListFields</td>
<td>(InputValue, wildCard, fieldType) Returns a list of fields found in the input value</td>
</tr>
<tr>
<td>ListIndexes</td>
<td>(InputValue, wildCard) Returns a list of attribute indexes found in the input value</td>
</tr>
<tr>
<td>ListDatasets</td>
<td>(wildCard, featureType) Returns the datasets in the current workspace</td>
</tr>
<tr>
<td>ListFeatureClasses</td>
<td>(wildCard, featureType) Returns the feature classes in the current workspace</td>
</tr>
<tr>
<td>ListRasters</td>
<td>(wildCard, rasterType) Returns a list of rasters found in the current workspace</td>
</tr>
<tr>
<td>ListTables</td>
<td>(wildCard, tableType) Returns a list of tables found in the current workspace</td>
</tr>
<tr>
<td>ListWorkspaces</td>
<td>(wildCard, workspaceType) Returns a list of workspaces found in the current workspace</td>
</tr>
</tbody>
</table>
Batch Processing

• Such a common programming structure
  – Sequence of operations
  – Iterated over list of items
  – Geoprocessing requires the inputs and outputs be organized
Enumerations

• Some methods return enumerations:
  – List (object) of values without a known count and of any type

• Objects to be listed can be restricted by data types and name
Parameters

- Wild cards: We can restrict the objects and datasets to be inserted into the list by name using an asterisk:

  ```python
  myPolygonList = arcpy.ListFeatureClasses("C*")
  myPolygonList = arcpy.ListFeatureClasses("*community*")
  ```

- Type filters: Restrict returns by certain type keywords:

  ```python
  myPolygonList = arcpy.ListFeatureClasses("C" , "polygon")
  ```

- Input dataset values to restrict by items that are a part of a certain object:

  ```python
  myPolygonList = arcpy.ListFields(table,"C" , "Integer")
  ```
Type Filters

- List methods by default list objects of all types
- If you specify a keyword, you restrict the list returned to those types

<table>
<thead>
<tr>
<th>Method</th>
<th>Type Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListDatasets</td>
<td>All, Feature, Coverage, RasterCatalog, CAD, VPF, TIN, Topology</td>
</tr>
<tr>
<td>ListFeatureClasses</td>
<td>All, Point, Label, Node, Line, Arc, Route, Polygon, Region</td>
</tr>
<tr>
<td>ListFields</td>
<td>All, SmallInteger, Integer, Single, Double, String, Date, OID, Geometry, BLOB</td>
</tr>
<tr>
<td>ListWorkspaces</td>
<td>All, Coverage, Access, SDE, Folder</td>
</tr>
<tr>
<td>ListTables</td>
<td>All, dBASE, INFO</td>
</tr>
<tr>
<td>ListRasters</td>
<td>All, ADRG, BIL, BIP, BSQ, BMP, CADRG, CIB, ERS, GIF, GIS, GRID, STACK, IMG, JPEG, LAN, SID, SDE, TIFF, RAW, PNG, NITF</td>
</tr>
</tbody>
</table>
Iterating Through Enumerated Features

- In pre-10.0, the “gp.” Geoprocessor Enumerated features are not Python lists

- So we iterate through them using while loops, not as for item in list: loops.

- You test the condition before the loop, then the loop iterates until the test results in null/false
Iterating Through Enumerations (cont’d.)

- Two methods and *no properties* for pre-10.0 enumerations:
  - `list.reset()` : Points to the top of the stack of objects, makes sure the first element is on top
  - `list.next()` : returns the currently selected value and increments the list index
import arcgisscripting
gp = arcgisscripting.create()
gp.workspace = "C:/tmp/Workspace"

try:
    themes = gp.ListFeatureClasses("*", 'All')
    themes.reset()
    theme = themes.next()

    while theme:
        print "\n","Theme name is: ", theme
        print theme, " contains the following field
        fields = gp.listfields(theme)
        fields.reset()
        field = fields.next()
        while field:
            print field.name
            field = fields.next()
        theme = themes.next()

except:
    print "WARNING: Some error occurred..."
    print gp.getmessages(0)
```python
import arcpy

arcpy.env.workspace = "H:/tmp/Week 10/data/"

try:
    feature_class_list = arcpy.ListFeatureClasses("*", "All")

    for feature_class in feature_class_list:
        print("\n"+"Theme name is: " + feature_class)
        print(feature_class + " contains the following fields: ")

        fields = arcpy.ListFields(feature_class)

        for field in fields:
            print(field.name)

except:
    print("WARNING: Some error occurred...")
    print(arcpy.GetMessages(0))
```
Listing Tools and Their Usage

```python
import arcpy

# Create a list of the default set of tools:
tools = arcpy.ListTools()

# Loop through the list and print each tool's usage:
for tool in tools[1:5]:
    print("/n Tool "+tool+": "+arcpy.Usage(tool))
```
import arcpy

arcpy.env.workspace = "H:/tmp/Week 10/data/

fcs = arcpy.ListFeatureClasses("*", "polygon")

# While the fc name is not empty, concatenate is to string:
inputs = ""
for fc in fcs:
    inputs += fc + "; "

print(inputs)

# Union the feature classes:
arcpy.Union_analysis(inputs, "union_output.shp")
Interactive Python in ArcGIS
Parser
- VB Script
- Python

Fields:
- FID
- Shape
- GRIDCODE
- ADMINID
- ADMINPOP
- YEARPOP
- ISO
- DIS_ID
- random

Type:
- Number
- String
- Date

Functions:
- .conjugate()
- .denominator()
- .imag()
- .numerator()
- .real()
- .as_integer_ratio()
- .fromhex()
- .hex()
- .is_integer()
- math.acos()
- math.acosh()

Pre-Logic Script Code:
```python
import random
def field_gauss(mean, sd):
    return random.gauss(mean, sd)

random = field_gauss(!ADMINPOP!, !ADMINPOP!/5)
```
Wrap-up

• The arcpy geoprocessing framework offers the gateway to ArcGIS’ tools

• It also provides functionality for sophisticated interactions with data
  – Batching is the most simple way to interact

• Python’s native utility (e.g. string manipulation) and extensible modules (charts, statistics) can be married with ArcGIS
In Lab This Week

• Explore Geoprocessing framework
  – Build simple models/tools
  – Export to scripts
  – Create batch processes

    import arcpy
    help(arcpy)

• Supplemental reading