Week 9 Lecture:
Data and Batch Geoprocessing With ArcGIS

Introduction to Programming for GIS & Remote Sensing
GEO6938-4172
GEO4938-4166
Reminder

This week, participate in the discussion forum

Practice problems are posted

Submit the “boxes” lab

Think about your proposed problems, data needs and goals for your project
Re-cap From Past Lectures

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

All this leads to one thing:
Code Re-use
The goal is to make life easier for us!

Imports
- Single functions
- Modules
- Classes/objects
- Entire processing frameworks:
  - PyGame, and ArcGIS’ Geoprocessing!
Objectives This Week

Learn about ArcGIS’ Geoprocessing objects

Building Models and integrating with scripts

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

An “ArcObjects” component

– Really just a single object with many, many sub-components

Allows accessing geoprocessing tools as native methods
Accessing 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)
```
The 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!

See the diagram again to explore that functionality…

– Cataloguing
– Describing
– Listing
– Editing
Class diagram on steroids...
The Geoprocessor Acts as Gateway…

To all of the power of ArcGIS and its tools:

Geoprocessor Programming Model
Reading the Programming Model

The geoprocessor object can be created in two ways. arcgiscripting can be used cross-platform. GpDispatch is limited to Windows operating systems.

**arcgiscripting / GpDispatch**

- Use CreateObject to create an object that may be used as a tool or method parameter.

**Properties**
- MaxSeverity
- MessageCount
- OverwriteOutput: Boolean
- ParameterCount
- Toolbox
  - AddError (Message)
  - 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, ExtraArg): Object

**FieldInfo**
- Count
  - AddField (FieldName, NewName, Visible, SplitRule)
  - ExportToOString()
  - FindFieldByName (FieldName)
  - GetFieldByNewName (FieldName)
  - GetFieldName (Index)
  - GetNewName (Index)
  - GetSplitRule (Index)
  - GetVisible (Index)
  - LoadFromString (InputString)
  - RemoveField (Index)
  - SetField (Index, FieldName)
  - SetNewName (Index, NewName)
  - SetSplitRule (Index, SplitRule)
  - SetVisible (Index, Visible)

**Projected Coordinate System only**
- CentralMeridian
- CentralMeridianInDegrees
- LongitudeOfOrigin
- LatitudeOf1st
- LatitudeOf2nd
- FalseEasting
- FalseNorthing
- CentralParallel1
- CentralParallel2
- StandardParallel1
- StandardParallel2
- LongitudeOf1st
- LongitudeOf2nd
- ScaleFactor
- Azimuth
- Classification
- PCSName
- PCSCode
- ProjectionName

**ValueTable***
- RowCount
- ColumnCount
- AddRow (optional value)
- GetRow (RowIndex)
- GetValue (RowIndex, ColumnIndex)
- LoadFromString (InputString)
- ExportToString
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 Geoprocessor’s List methods
– Use for different data types
– List objects for iteration

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListFields</td>
<td>Returns a list of fields found in the input value</td>
</tr>
<tr>
<td>ListIndexes</td>
<td>Returns a list of attribute indexes found in the input value</td>
</tr>
<tr>
<td>ListDatasets</td>
<td>Returns the datasets in the current workspace</td>
</tr>
<tr>
<td>ListFeatureClasses</td>
<td>Returns the feature classes in the current workspace</td>
</tr>
<tr>
<td>ListRasters</td>
<td>Returns a list of rasters found in the current workspace</td>
</tr>
<tr>
<td>ListTables</td>
<td>Returns a list of tables found in the current workspace</td>
</tr>
<tr>
<td>ListWorkspaces</td>
<td>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 = gp.ListFeatureClasses("C*")
myPolygonList = gp.ListFeatureClasses("*community*")
```

Type filters: Restrict returns by certain type keywords:

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

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

```python
myPolygonList = gp.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

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 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"
	ry:
    themes = gp.ListFeatureClasses("*", 'All')
    themes.reset()
    theme = themes.next()
    while theme:
        print "\n","Theme name is: ", theme
        print theme, " contains the following fields:
        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)
import arcgis scripting

# Create the GeoProcessor
gp = arcgis scripting.create()

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

# Get the first tool name from the list.
tools.Reset()
tool = tools.Next()

# Loop through the list and print each tool.
while tool:
    print gp.Usage(tool)
tool = tools.Next()
import arcgisscripting

# Create the GeoProcessor
gp = arcgisscripting.create()
gp.Workspace = "C:/tmp/Workspace"

fcs = gp.ListFeatureClasses("*", "polygon")
fcs.reset()
fc = fcs.next()  # Get first feature class name
inputs = ""  # Start the string variable to contain features to Union

while fc:  # While the fc name is not empty, concatenate is to string
    inputs += "; " + fc
    fc = fcs.next()

print inputs

# Union the feature classes
gp.union_analysis(inputs, "union_output.shp")

# Or union specific feature classes and only carry the FID attributes to output
##gp.union_analysis("watersheds.shp; watersheds_sub.shp", "union_output.shp", "O
Wrap-up

The 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

Supplemental reading: