JANUARY 12, 2017

USING THE PYTHON WINDOW EDITOR

TOPIC 7

VINCENT A. DINOTO, JR.
JEFFERSON COMMUNITY AND TECHNICAL COLLEGE
Vince.dinoto@kctcs.edu

Contents

Table of Figures

Introduction

In this module, the use of the Python Window that is contained within Esri ArcMap Desktop will be covered. It is assumed that the learner is using version 10.1 or higher of the software. The author constructed this lesson-using version 10.3 and higher of Esri ArcMap. The interfaces for the older version should look similar.

The Python window is a smart immediate mode editor, which allows for saving information. Thus on the pressing of Enter or Return, the script line will execute. When a colon is the last character of functional code it does not execute the code until the loop is completed. Loops and decision scripts will be executed by pressing Enter twice on the keyboard.

Functions

- Use of the ArcMap Python Window
- Buffering
- Merging
- Spatial reference (determination)
- GetCount
- List functions

Getting Started

Figure 1: Python Window

The Python Window is opened from the control ribbon in ArcMap next to the Toolbox. The Window can be pinned or left to float, if working with dual monitors; it is suggested to put the Python Window on the second monitor, see Figure 1.

The Python Window is similar to an IDE, but it is not required that arcpy be loaded since the operation is being done within ArcMap and by default, it is already loaded. There are no issues with the script if it contains loading statements for arcpy. Generally retaining those lines of code makes the scripts more universal in how they function. Therefore, these scripts will also run outside of the ArcMap environment.

Code is saved by right clicking in the window and selecting save. Loading code works in the same general way. Care must be taken in the saving of code since whatever is visible in the Python Window will be saved which could include errors. It is generally not recommend saving the code from this Window, but loading the code into this window can be useful.

To understand how the Python Window functions, the first example will use code that was covered earlier in the course. In this way, the learner can explore the similarities and the differences. As the code is typed in, the Python window suggestions are made to the formatting of the tool/command, which can save time in the input steps. All code covered in this module could also work directly in an IDE, copy and paste functions can be used as well.

Buffering

This example was done previously in the buffer discussion. The learner may wish to repeat the procedure in the IDE so that a comparison on the similarities and differences can be made.

Figure 3 shows the raw data prior to inputting the script. In the first line of the code in Figure 3, the environment module is imported. The next line of code sets the workspace, so only the file names are required and not a complete pathway, the learner's pathway will be different. The names of the input and output files are defined as string variables. If the name of the output file is still contained within the geodatabase from when the Buffering lesson was completed, that file name must be removed or a different name used. The code will not over write an existing file. The radius is defined as another variable and it is a string, it must be a string since the variable contains the units that are required. The final line of the script is the functional line of code, which is executing an operational buffer command. The results of the buffer command completed a 25-mile radius around each point with no dissolve; see Figure 4.

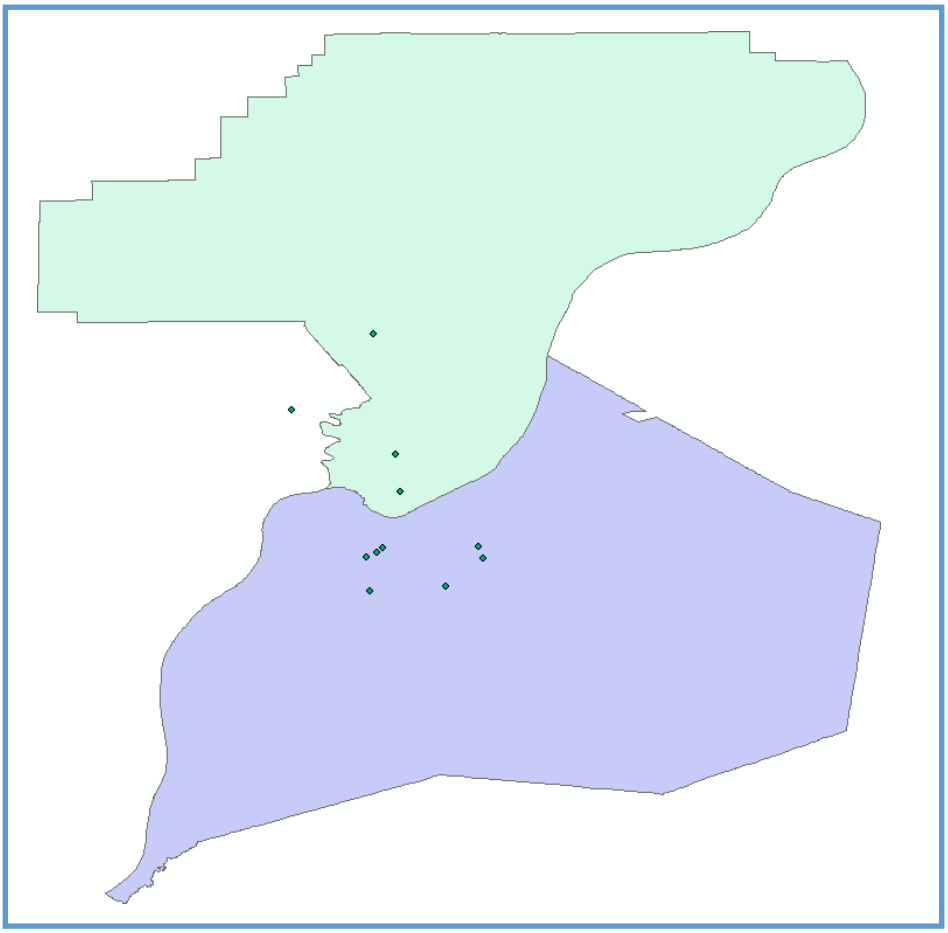

Figure 2: Clark and Jefferson Counties and selected points.

```
>>> from arcpy import env
>>> env.workspace="x:\users\vdinotojr0001\gis data\ky1.gdb"
>>> featureInput=r"colleges"
>>> featureOutput=r"college_buffer"
>>> radius="25 miles"
>>> arcpy.Buffer_analysis(featureInput,featureOutput,radius)
```

Figure 3: Buffering Script

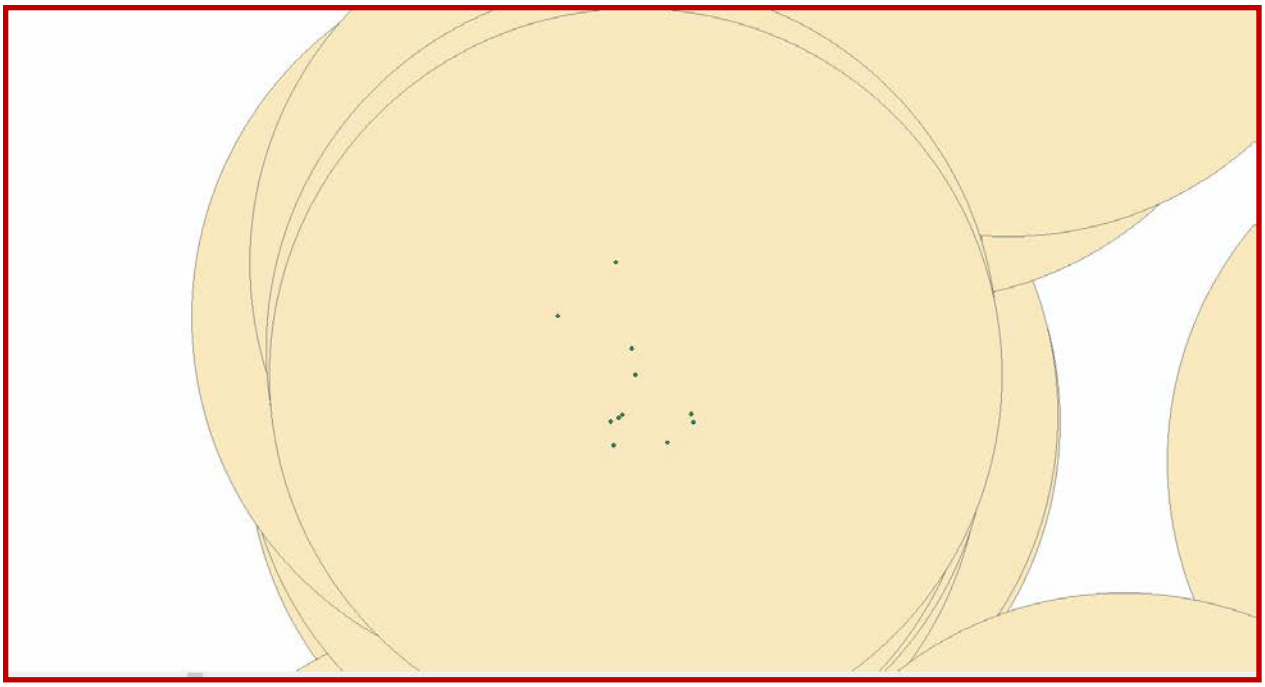

Figure 4: Buffer results

Merging

```
>>> from arcpy import env
>>> env.workspace="x:\users\vdinotojr0001\gis data\ky1.gdb"
>>> listCounty=["KY_Jefferson","IN_Clark"]
>>> arcpy.Merge_management(listCounty, "Clark_Jefferson")
<Result 'C:\\Users\\vdinotojr0001\\Documents\\ArcGIS\\Default.gdb\\Clark_Jefferson'>
```

Figure 5: Merge Script

The Merge command was covered previously, but the example shown in Figure 5 is slightly different.

The first two lines of the code are setting up the environment, so that shorter names can be used in the actual operational parameters. The third line of code creates a list variable, representing the names of the files to be merged. The fourth line is the actual merge operation, which will

merge the members of the list together and create a new file, which will be named Clark_Jefferson.. The final line is the result of the operation\

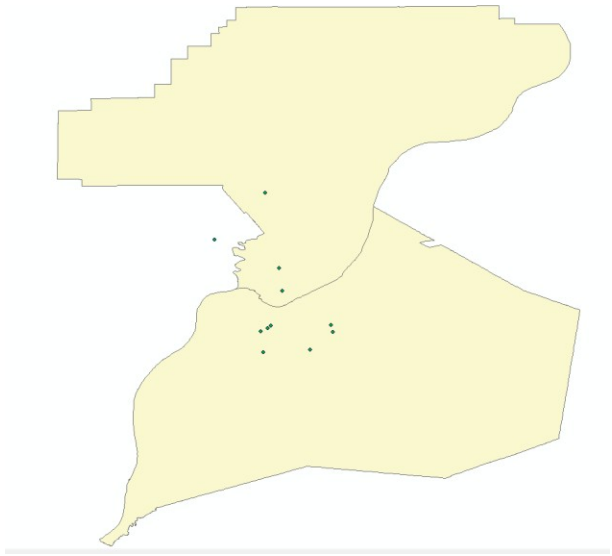

Figure 6: Result of the Merge

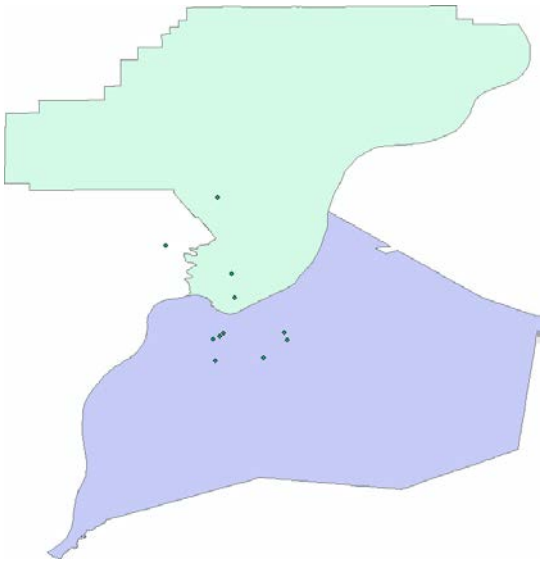

Figure 7: Before the Merge

Assignment 7.1

The assignment requires merging five counties. The merge will be used as the clipping surface for the college file. This should be done in the ArcMap Python Window. Provide your instructor with the results of the clipping operation, both before and after the operation as well as the code, so that it can be determined that it was completed using ArcMap.

Spatial Reference

Many times in geospatial work, the user needs to know the projection of different files in which they are working with, which generally requires a manual inspection of each file. This is a time consuming process especially when there are many files contained within the project. If the file is not loaded and not used in a geodatabase then a manual inspection of the projection file is required using Windows Explorer. The following script will utilize a new command call **SpatialReference**. This command can be placed in a loop to determine the spatial reference of all files in a mapping project, in this case, only a single file is used; therefore, it is probably quicker to do this manually. The code if Figure 8 assumes the file is in a folder and not a geodatabase.

```
>>> projection="Y:\Vince\Colleges.prj"
>>> #define the variable (the layer)
>>> spatial=arcpy.SpatialReference(projection)
>>> filePrj=spatial.name
>>> print filePrj
USA_Contiguous_Albers_Equal_Area_Conic
>>>
```

Figure 8: Spatial Reference Script

- The first line of the code in Figure 8 sets the pathway to a projection shapefile. The pathway will be different on the learner's computer.
- The next functional line of code defines a new variable based on the **SpatialReference** operation.
- A new variable is created which will contain only the name of what was retrieved with the **SpatialReference** command. The reason this new variable was required is that the **SpatialReference** command returns more information than just the name.
- The final operation is to print the projection of the file.

The final text line in Figure 8 shows the resulting projection, which is shown in gray. This methodology could be used if the file is open or if closed. This operation could have been completed in an IDE, but would require loading the arcpy module.

Get Count

Typically when a user wants to know the number of elements in a shapefile it requires loading it into ArcMap, opening the attribute table and looking at the number of records. The **GetCount** command will provide the result in a single line of code. The script can be run in immediate mode inside or outside of ArcMap, if run from outside of ArcMap the arcpy module and pathways will need to be established. The results shown in Figure 9 is for the College layer already loaded on the map from the previous example.

```
>>> arcpy.GetCount_management("Colleges")
<Result '3766'>
>>>
```

Figure 9: Get Count

List Function

There are several different types of list functions and some of them will be explored in this section.

List Feature Class

In the first example, a simple function will be shown which will list all the members contained within a geodatabase or a folder. This type of list is known as a feature class. **ListFeatureClass** requires that an environment pathway be set prior to using the command.

```
>>> from arcpy import env
>>> env.workspace="Y:\Vince"
>>> feature=arcpy.ListFeatureClasses()
>>> print feature
[u'Geocoding_Result.shp', u'Colleges.shp']
>>>
```

Figure 10: List Feature Class version 1

In Figure 10 the members of a folder will be displayed. The folder is named Vince and located on the author's Y drive. The first two lines of code set the environment workspace to the folder location. It is important to note the direction of the slashes for the pathway to the folder. The third line sets the value of the variable '***feature***' to the members in the workspace by default, since there are no values placed in the parentheses. The default is a list of all members of the

folder. Parameters could be placed in the parentheses, which could look for values of a single state or only a specific type of file such as a point file.

```
>>> env.workspace="X:/Users/vdinotojr0001/gis data/KY1.gdb"
>>> feature=arcpy.ListFeatureClasses("IN*")
>>> print feature
[u'IN_Clark', u'IN_KY_Merge', u'IN_KY_Merge_Census', u'IN_Cla
>>>
```

Figure 11: List Feature Class version 2

In the example, shown in Figure 11, the members of a geodatabase will be explored as well as searching for only specific members of the geodatabase. This is not a relational database that is being searched.

1. It is important to note in the setting of the workspace that the slashes are going in the opposite direction then those in the previous example (the pathway would be different for the learner).
2. In the second line of code that a wildcard search has been added between the parentheses. The asterisk shows that as long as IN is located in the search a result will be returned. IN represents the state of Indiana. Therefore, if a good naming convention was used within the geodatabase, the returned results will be all files that are related to IN in the geodatabase. Note the geodatabase has not been properly named since its name is KY for Kentucky (but does contain items from Indiana).
3. The variable ***feature***, which is the result of the command, and then printed
4. The results are shown in gray.
   In Figure 11 the results line has been clipped for display purposes.

List Fields

ListFields is a new function, but has similarieties to the **ListFeatureClasses**, except it is dealing with the fields inside of a single file. For example; if the fields names contained within a shapefile are needed, typically the user would need to load the shapefile into ArcMap, then open the attribute table to discover the names. Other parameters can also be returned using this command, but for this example only the field names will be returned. In this example a **For** statement will be used so that all fields contained within the layer are displayed. The output is shown in a gray color in Figure 13.

1. The first line shown in Figure 12 is to set the environment just as was done in Figure 10. Since the shapefile to be explored is located in a folder this code is used. If the shapefile had been in a geodatabase, the pathway would be written differently.
2. The next functional line of code is creating a list variable (***fieldName***) based on the

```
>>> env.workspace="X:\Users\vdinotojr0001\gis data"
>>> fieldName=arcpy.ListFields("Colleges")
>>> for field in fieldName:
...     print field.name
...
```

Figure 12: List Fields Script

```
FID
Shape
Status
Score
Match_type
Match_addr
Side
AddNum
AddNumFrom
AddNumTo
StPreDir
StPreType
StName
StType
StDir
Nbrhd
City
Subregion
```

Figure 13: Results of ListFields

contents of the College shapefile, which is contained in the gis data folder.

3. The '***fieldName'*** variable will contain more information than just the name of the field.Only the name is required, which is specified in the **print** statement. This is accomplished using the word '***name'*** with the variable ***field***.
4. The shapefile that is being used has numerous attributes therefore, a **for** statement is used to insure that each attribute is displayed as an individual line. The **For** loop will be repeated until all the elements in **fieldName** are printed.

To execute the operation it requires pressing the enter key twice since there was a loop used in the operation. The print line has a different look from the lines that were executed when the enter key was pressed.

Assignment 7.2

In this assignment, a code will be written in the Python Window of ArcMap that will return the fields in three different shapefiles that begin with the letter F. This will require the learner to use loops and both of the list functions that were discussed. The learner will use list variables since multiple items will be returned. Provide your instructor with the script file and a screen image of the results and the code.

Conclusion

It is important to keep in mind that the Python Window in ArcMap works similarly to the immediate mode in an IDE. The Python script window will give hints as commands are being typed on the format and the command names. The files used in ArcMap, in general, can be loaded, but this is not a requirement. Realize that some operations cannot be completed on an opened file due to locks installed by the software. A lock is a file protection so that the file cannot be modified while opened, thus two users cannot be editing at the same time, and generally, this can be done only with a relational database. The database operations discussed are for personal geodatabases and might function differently if used with a relational database. The separating slashes in pathways in the environment statement are different for folders and for geodatabases.

Index

This workforce product was funded by a grant awarded by the U.S. Department of Labor's Employment and Training Administration. The product was created by the grantee and does not necessarily reflect the official position of the U.S. Department of Labor. The U.S. Department of Labor makes no guarantees, warranties, or assurances of any kind, express or implied, with respect to such information, including any information on linked sites and including, but not limited to, accuracy of the information or its completeness, timeliness, usefulness, adequacy, continued availability, or ownership. This is an equal opportunity program. Assistive technologies are available upon request and include Voice/TTY (771 or 800-947-6644).